

BaseX Documentation

Version 7.2.1

Contents

Articles

Main Page	1
-----------	---

Getting Started **3**

Getting Started	3
Startup	4
Startup Options	6
Start Scripts	13

User Interfaces **17**

Graphical User Interface	17
Shortcuts	21
Database Server	24
Standalone Mode	27
Web Application	28

General Info **31**

Databases	31
Binary Data	33
Parsers	34
Commands	38
Options	51

Integration **65**

Integrating oXygen	65
Integrating Eclipse	67

Query Features **69**

XQuery	69
XQuery 3.0	70
Higher-Order Functions	75
Module Library	82
Repository	83
Java Bindings	88
Full-Text	91
Full-Text: Japanese	94

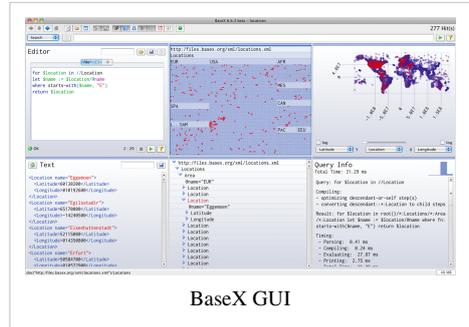
XQuery Update	96
Serialization	100
XQuery Errors	102
XQuery Modules	114
Cryptographic Module	114
Database Module	119
File Module	127
Full-Text Module	132
HTTP Module	135
Higher-Order Functions Module	138
Index Module	140
JSON Module	142
Map Module	147
Math Module	151
Repository Module	155
SQL Module	157
Utility Module	160
XSLT Module	165
ZIP Module	168
ZIP Module: Word Documents	170
Developing	172
Developing	172
Integrate	173
Git	174
Maven	180
Releases	182
Translations	183
HTTP Services	184
REST	184
REST: POST Schema	192
RESTXQ	193
WebDAV	197
WebDAV: Windows 7	198
WebDAV: Windows XP	200
WebDAV: Mac OSX	203

WebDAV: GNOME	205
WebDAV: KDE	207
Client APIs	209
Clients	209
Standard Mode	210
Query Mode	211
PHP Example	213
Server Protocol	214
Server Protocol: Types	218
Java Examples	220
Advanced User's Guide	222
Advanced User's Guide	222
Configuration	223
Indexes	224
Backups	226
Catalog Resolver	227
Statistics	229
Storage Layout	233
Node table storage	235
User Management	236
Transaction Management	237
Logging	239
Events	240
Execution Plan	242
References	
Article Sources and Contributors	243
Image Sources, Licenses and Contributors	245
Article Licenses	
License	246

Main Page

Welcome to the documentation of BaseX!

BaseX ^[1] is both a light-weight, high-performance and scalable XML Database and an XPath/XQuery Processor with full support for the W3C Update and Full Text extensions. It focuses on storing, querying, and visualizing large XML and JSON documents and collections. A visual frontend allows users to interactively explore data and evaluate queries in realtime (i.e., with each key click). BaseX is platform-independent and distributed under the free BSD License (find more in Wikipedia ^[2]).



BaseX GUI

This documentation is based on **BaseX 7.2.1**. It can also be downloaded (BaseX72.pdf ^[3]). Features that have recently been changed are flagged in red (e.g.: [Version 7.2.2](#)).

Getting Started

The Getting Started Section gives you a quick introduction to BaseX. We suggest you to start with the Graphical User Interface as this is the easiest way to access your XML data, and to get an idea of how XQuery and BaseX works.

Categories: Beginners

XQuery Portal

More information on using the wide range of XQuery functions and performing XPath and XQuery requests with BaseX can be found in our XQuery Portal.

Categories: XQuery

Developer Section

The Developer Section provides useful information for developers. Here you can find information on our supported client APIs and HTTP services, and we present different ways how you can integrate BaseX into your own project.

Categories: Developer, HTTP, API

Advanced User's Guide

Information for advanced users can be found in our Advanced User's Guide, which contains details on the BaseX storage, the Client/Server architecture, and some querying features.

Categories: Internals

You are invited to contribute to our Wiki: it's easy to get a new account.

If you have questions and are looking for direct contact to developers and users, please write to our mailing list ^[4] (basex-talk@mailman.uni-konstanz.de).

References

- [1] <http://basex.org>
 - [2] <http://en.wikipedia.org/wiki/BaseX>
 - [3] <http://files.basex.org/releases/7.2/BaseX72.pdf>
 - [4] <http://basex.org/open-source/>
-

Getting Started

Getting Started

This page is one of the Main Sections of the documentation. It gives a quick introduction on how to start, run, and use BaseX.

Getting Started

- Startup: How to get BaseX running
- Startup Options

User Interfaces

- Graphical User Interface (see available Shortcuts)
- Database Server: The client/server architecture
- Standalone Mode: The comand-line interface
- Web Application: The HTTP server

General Info

- Databases: How databases are created, populated and deleted
- Parsers: How different input formats can be converted to XML
- Commands: Full overview of all database commands
- Options: Listing of all database options

Integration

- Integrating oXygen
 - Integrating Eclipse
-

Startup

This article is part of the Getting Started Guide. It tells you how to get BaseX running.

Requirements

BaseX

Please download ^[1] the latest BaseX version from our homepage. The official releases include the BaseX JAR file, libraries and optional Start Scripts. If you do not use an installer, we recommend to manually add the project's `bin` directory to your path environment; this way, you will be able to run BaseX from everywhere in your shell/terminal.

Java

A Runtime Environment of Java 1.6 ^[2] (JRE) is needed to run BaseX. BaseX is platform independent and runs on any system that provides a Java Virtual Machine. BaseX has been tested on Windows (2000, XP, Vista, 7), Max OS X (10.x), Linux(SuSE xxx, Debian, Ubuntu) and OpenBSD (4.x).

Synchronization

If you plan to concurrently read and write your data, you need to resort to the client/server architecture. This is because the GUI, the standalone client and the client/server architecture of BaseX all work in different JVMs (Java virtual machines) and are not synchronized among each other. You should be aware that, in the worst case, databases might get corrupt if you use BaseX instances in different JVMs and try to enforce updates.

BaseX GUI

The GUI is the visual interface to the features of BaseX. It can be used to create new databases, perform queries or interactively explore your XML data.

The GUI can be started as follows (get more information on all Startup Options):

- Double click on the `BaseX.jar` file.
- Run one of the `basexgui` or `basexgui.bat` scripts.
- Execute the following command: `java -cp BaseX.jar org.baseX.BaseXGUI`
- On *Windows*: Double click on the **BaseX GUI** icon.
- For Maven users: type in `mvn exec:java` in the main directory of the `baseX` project.

BaseX Standalone

The Standalone Mode can be used to enter database commands, or execute automated database operations and queries, on command line.

The standalone version can be started as follows (get more information on all Startup Options):

- Run one of the `baseX` or `baseX.bat` scripts.
 - Execute the following command: `java -cp BaseX.jar org.baseX.BaseX`
 - On *Windows*: Double click on the **BaseX** icon.
-

BaseX Server

The Database Server comes into play if BaseX is to be used by more than one user (client). It handles concurrent read and write transactions, provides user management and logs all user interactions.

By default, the server listens to the port 1984. There are several ways of starting and stopping the server (get more information on all Startup Options):

- Run one of the `basexserver` or `basexserver.bat` scripts. Add the `stop` keyword to gracefully shut down the server.
- Execute the following command: `java -cp BaseX.jar org.baseX.BaseXServer`. Again, the `stop` keyword will ensure a graceful shutdown.
- On *Windows*: Double click on the **BaseX Server** icon, which will also start the HTTP Server, or the **BaseX Server (stop)** icon.

Pressing `Ctrl+c` will close all connections and databases and shut down the server process.

BaseX Client

The BaseX Client interface can be used to send commands and queries to the server instance on command line.

It can be started as follows (get more information on all Startup Options):

- Run one of the `basexclient` or `basexclient.bat` scripts.
- Execute the following command: `java -cp BaseX.jar org.baseX.BaseXClient`
- On *Windows*: Double click on the **BaseX Client** icon.

The default `admin` user can be used to connect to the server:

- **Username:** `admin`
- **Password:** `admin`

The password should be changed with the `PASSWORD` command after the first login.

Please check out the article on the Database Server for more details.

BaseX HTTP Server

The HTTP Server gives access to the REST, RESTXQ and WebDAV Services of BaseX. By default, it starts an instance of the Jetty Web Server^[3], which listens to the port 8984, and the BaseX Server, which listens to 1984.

The HTTP Server can be started as follows (get more information on all Startup Options):

- Run one of the `basexhttp` or `basexhttp.bat` scripts. Add the `stop` keyword to gracefully shut down the server.
- On *Windows*: Double click on the **BaseX Server** or **BaseX Server (stop)** icon.
- For Maven users: type in `mvn jetty:run` in the main directory of the `basex-api` project, and press `Ctrl+c` to shut down the process.
- You may also want to deploy BaseX as a Web Application.

Changelog

Version 7.0

- Updated: the BaseXJAXRX has been replaced with BaseXHTTP

References

- [1] <http://basex.org/download>
- [2] <http://www.java.com>
- [3] <http://jetty.codehaus.org/jetty/>

Startup Options

This article is part of the Getting Started Guide. It gives more details on the command-line options of all BaseX start scripts.

BaseX GUI

Launch the GUI

```
$ basexgui [file]
```

One or more XML and XQuery files can be passed on as parameters. If an XML file is specified, a database instance is created from this file, or an existing database is opened. XQuery files are opened in the XQuery editor.

BaseX Standalone

Launch the console mode

```
$ basex
BaseX [Standalone]
Try "help" to get more information.
> _
```

Available command-line flags can be listed with `-h`:

```
$ basex -h
BaseX [Standalone]
Usage: basex [-bcCdiLoqsuvVwxz] [file]
  [file]      Execute XQuery file
  -b<pars>   Bind external XQuery variables
  -c<cmds>   Execute database commands
  -C<file>   Execute commands from script
  -d         Activate debugging mode
  -i<input>  Open initial file or database
  -L         Add trailing newline after query result
  -o<file>   Write output to file
  -q<expr>   Execute XQuery expression
  -s<pars>   Set serialization parameter(s)
  -u         Write updates back to original files
```

-v/V	Show (all) process info
-w	Preserve whitespaces from input files
-x	Show query execution plan
-z	Skip output of results

The meaning of all flags is listed in the following. If an equivalent database option exists (which can be specified via the SET command), it is listed as well:

Flag	Description	Option	Examples
[file]	Executes the specified XQuery file.		
-b<pars>	Binds external variables to XQuery expressions. This flag may be specified multiple times. Variables names and their values are delimited by equality signs (=). The names may be optionally prefixed with dollar signs. If a variable uses a namespace different to the default namespace, it can be specified with the Clark Notation ^[1] or Expanded QName Notation ^[2] .	BINDINGS	<ul style="list-style-type: none"> -b\$v=example -q\$v -b{URL}ln=value -q"declare namespace ns='URL'; \$ns:ln"
-c<cmd>	Runs the specified commands and quits. Several commands can be separated by semicolons.		-c"list;info"
-C<file>	Runs all commands from the specified text file and quits. Empty lines and lines starting with the number sign # are skipped.		-C commands.txt
-d	Turns on the debugging mode. Debugging information is output to <i>standard error</i> .	DEBUG	
-i<input>	Opens a database or XML document specified by the argument. The opened input may be further processed by an XQuery expression.		-iitems.xml -q"//item"
-L	Add trailing newline after query result. Useful for batch processing or pipelining on Unix systems.		
-o<file>	All command and query output is written to the specified file.		-o output.txt
-q<expr>	Executes the specified string as XQuery expression.		-q"doc('input')//head"
-s<pars>	Specifies parameters for serializing XQuery results; see <i>Serialization</i> for more details. This flag may be specified multiple times. Key and values are separated by the equality sign (=).	SERIALIZER	-smethod=text
-u	Modifies original files after evaluating XQuery Update expressions.	WRITEBACK	
-v	Prints process and timing information to the <i>standard output</i> .		
-V	Prints detailed query information to the <i>standard output</i> , including details on the compilation and profiling steps.	QUERYINFO	
-w	By default, whitespaces around text nodes are chopped when a database is created. This flag can be specified to preserve whitespaces.	CHOP	
-x	This flags turn on the output of the query execution plan, formatted in XML.	XMLPLAN	
-z	Skips the serialization of XQuery results. This flag is useful if the query is profiled or analyzed.	SERIALIZE	

Multiple query files and `-c/-i/-q` flags can be specified. All other options will be set before any other operation takes place. The specified inputs, query files, queries and commands will be subsequently evaluated after that in the given order. The standard input can be parsed by specifying a single dash (-) as argument.

BaseX Server

Launch the server

```
$ basexserver
BaseX [Server]
Server was started.
```

Available command-line flags can be listed with `-h`:

```
$ basexserver -h
BaseX [Server]
Usage: basexserver [-cdeipSz] [stop]
  stop      Stop running server
  -c<cmds> Execute initial database commands
  -d        Activate debugging mode
  -e<num>   Set event port
  -i        Enter interactive mode
  -p<num>   Set server port
  -S        Start as service
  -z        Suppress logging
```

The flags have the following meaning (equivalent database options are shown in the table as well):

Flag	Description	Option	Default	Examples
<code>stop</code>	Stops an existing server instance and quits.			
<code>-c<cmd></code>	Launches database commands before the server itself is started. Several commands can be separated by semicolons.			<code>-c"open database; info"</code>
<code>-d</code>	Turns on the debugging mode. Debugging information is output to <i>standard error</i> .	DEBUG		
<code>-e<num></code>	Specifies the port on which the server will send events to clients.	EVENTPORT	1985	<code>-e9998</code>
<code>-i</code>	Starts the interactive console mode, which can be used to enter database commands. This mode is similar to the default standalone and client mode.			
<code>-p<num></code>	Specifies the port on which the server will be addressable.	PORT	1984	<code>-p9999</code>
<code>-S</code>	Starts the server as service (i.e., in the background).			
<code>-z</code>	Does not generate any log files.			

Multiple `-c` and `-i` flags can be specified. All other options will be set before any other operation takes place. The specified inputs, query files, queries and commands will be subsequently evaluated after that in the given order. The standard input can be parsed by specifying a single dash (`-`) as argument.

BaseX Client

Launch the console mode communicating with the server

The user name and password will be requested. The default user/password combination is **admin/admin**:

```
$ basexclient
Username: admin
Password: *****
BaseX [Client]
Try "help" to get more information.
> _
```

Available command-line flags can be listed with `-h`:

```
$ basexclient -h
BaseX [Client]
Usage: basexclient [-bcCdiLnopQqsUvVwxz] [file]
 [file]      Execute XQuery file
 -b<pars>   Bind external XQuery variables
 -c<cmds>   Execute database commands
 -C<file>   Execute commands from script
 -d         Activate debugging mode
 -i<input>  Open initial file or database
 -L         Add trailing newline after query result
 -n<name>   Set server (host) name
 -o<file>   Write output to file
 -p<num>    Set server port
 -P<pass>   Specify user password
 -q<expr>   Execute XQuery expression
 -s<pars>   Set serialization parameter(s)
 -U<name>   Specify user name
 -v/V       Show (all) process info
 -w         Preserve whitespaces from input files
 -x         Show query execution plan
 -z         Skip output of results
```

The flags have the following meaning (equivalent database options are shown in the table as well):

Flag	Description	Option	Default	Examples
[file]	Executes the specified XQuery file.			
-b<pars>	Binds external variables to XQuery expressions. This flag may be specified multiple times. Variables names and their values are delimited by equality signs (=). The names may be optionally prefixed with dollar signs. If a variable uses a namespace different to the default namespace, it can be specified with the Clark Notation ^[1] or Expanded QName Notation ^[2] .	BINDINGS		<ul style="list-style-type: none"> -b\$v=example -q\$v -b{URL}ln=value -q"declare namespace ns='URL'; \$ns:ln"
-c<cmd>	Runs the specified commands and quits. Several commands can be separated by semicolons.			-c"list;info"

-C<file>	Runs all commands from the specified text file and quits. Empty lines and lines starting with the number sign # are skipped.		-C commands.txt	
-d	Turns on the debugging mode. Debugging information is output to <i>standard error</i> .	DEBUG		
-i<input>	Opens a database or XML document specified by the argument. The opened input may be further processed by an XQuery expression.			-iitems.xml -q"//item"
-L	Add trailing newline after query result. Useful for batch processing or pipelining on Unix systems.			
-n<name>	Specifies the host name on which the server is running.	HOST	localhost	-nserver.basex.org
-o<file>	All command and query output is written to the specified file.			
-p<num>	Specifies the port on which the server is running.	PORT	1984	-p9999
-P<pass>	Specifies the user password. If this flag is omitted, the password will be requested on command line. <i>Warning:</i> when the password is specified via this flag, it may be visible to others.			-Uadmin -Padmin
-q<expr>	Executes the specified string as XQuery expression.			-q"doc('input')//head"
-s<pars>	Specifies parameters for serializing XQuery results; see <i>Serialization</i> for more details. This flag may be specified multiple times. Key and values are separated by the equality sign (=).	SERIALIZER		-smethod=text
-U<name>	Specifies the user name. If this flag is omitted, the user name will be requested on command line.			-Uadmin
-v	Prints process and timing information to the <i>standard output</i> .			
-V	Prints detailed query information to the <i>standard output</i> , including details on the compilation and profiling steps.	QUERYINFO		
-w	By default, whitespaces around text nodes are chopped when a database is created. This flag can be specified to preserve whitespaces.	CHOP		
-x	This flags turn on the output of the query execution plan, formatted in XML.	XMLPLAN		
-z	Skips the serialization of XQuery results. This flag is useful if the query is profiled or analyzed.	SERIALIZE		

Multiple query files and `-c/-i/-q` flags can be specified. All other options will be set before any other operation takes place. The specified inputs, query files, queries and commands will be subsequently evaluated after that in the given order. The standard input can be parsed by specifying a single dash (-) as argument.

BaseX HTTP Server

Launch the HTTP server

```
$ basexhttp
BaseX [Server]
Server was started.
2011-01-02 03:04:05.600:INFO::Logging to STDERR via org.mortbay.log.StdErrLog
2011-01-02 03:04:05.700:INFO::jetty-6.1.26
2011-01-02 03:04:05.800:INFO::Started SocketConnector@0.0.0.0:8984
```

Available command-line flags can be listed with `-h`:

```
$ basexhttp -h
BaseX [HTTP]
Usage: basexhttp [-dehlnpPRUWz] [stop]
  stop          Stop running server
  -c            Start in client mode
  -d            Activate debugging mode
  -e<num>      Set event port
  -h<num>      Set port of HTTP server
  -l            Start in local mode
  -n<name>     Set host name of database server
  -p<num>      Set port of database server
  -P<pass>     Specify user password
  -R            Deactivate REST service
  -s            Specify port to stop HTTP server
  -S            Start as service
  -U<name>     Specify user name
  -W            Deactivate WebDAV service
  -X            Deactivate RESTXQ service
  -z            Suppress logging
```

The flags have the following meaning (equivalent database options are shown in the table as well):

Flag	Description	Option	Default	Examples
stop	Stops a running HTTP server. By default, the database server will be stopped as well, unless one of the flags <code>-c</code> or <code>-l</code> is specified.			
-c	Starts the server in <i>client mode</i> , and sends all commands to a database server instance.			
-d	Turns on the debugging mode. Debugging information is output to <i>standard error</i> .	DEBUG		
-e<num>	Specifies the port on which the server will send events to clients.	EVENTPORT	1985	-e9998
-h<num>	Specifies the port on which the HTTP server will be addressable. This port is e.g. specified in the HTTP URLs.	HTTPPORT	8984	-h9999
-l	Starts the server in <i>local mode</i> , and executes all commands in the embedded database context.			
-n<name>	Specifies the host name on which the server is running.	HOST	localhost	-nserver.basex.org
-p<num>	Specifies the port on which the BaseX Server will be addressable.	SERVERPORT	1984	-p9998

-P<pass>	Specifies the user password, which will be used by the HTTP services to open a new session. If this flag is omitted, the password will be requested on command line. <i>Warning:</i> when the password is specified on command-line, it may be visible to others.		admin	-Uadmin -Padmin
-R	Deactivates the REST service.			
-s<num>	Specifies the port that will be used to stop the HTTP server.	STOPPORT		
-S	Starts the server as service (i.e., in the background).			
-U<name>	Specifies the user name, which will be used by the HTTP services for opening a new session. If this flag is omitted, the password will be requested on command line.		admin	-Uadmin
-W	Deactivates the WebDAV service.			
-X	Deactivates the RESTXQ service. <i>Since Version 7.2</i>			
-z	Does not generate any log files.			

Some options can also be triggered by setting the following system properties:

- `org.baseX.user`: user name for opening new sessions
- `org.baseX.password`: user password for opening new sessions
- `org.baseX.mode`: by default, the HTTP server also starts an instance of the database server. If the mode is set to `local`, an embedded database context is used for executing commands. If the mode is set to `client`, all commands are sent to an existing database server instance.
- `org.baseX.path`: path to the BaseX Home Directory

Changelog

Version 7.2

- Added: RESTXQ Service

Version 7.1.1

- Added: Options `-C` and `-L` in standalone and client mode

Version 7.1

- Multiple query files and `-c/-i/-q` flags can be specified.

References

- [1] <http://www.jclark.com/xml/xmlns.htm>
 [2] <http://www.w3.org/TR/xquery-30/#id-basics>

Start Scripts

The following scripts, which are mentioned on the Startup and Startup Options pages, are also included in our official **Windows** and **ZIP** release files, and available in our [GitHub repository](#) ^[1]:

- The installers automatically add the project's `bin` directory to your path environment.
- Otherwise, you can manually add the `bin` directory to your `PATH` environment variable.

BaseX Main Package

The following scripts launch the standalone version of BaseX. Please replace the class name in `org.basex.BaseX` with either `BaseXClient`, `BaseXServer`, or `BaseXGUI` to run the client, server or GUI version.

Windows: basex.bat

```
@setlocal
@echo off

REM Path to this script
set PWD=%~dp0

REM Paths to distributed JAR archives or class files
set BASEX=basex.jar

REM Options for virtual machine
set VM=-Xmx512m

REM Classpath
set LIB=%PWD%/../lib
set CP=%BASEX%;%LIB%/lucene-analyzers-3.0.2.jar;%LIB%/tagsoup-1.2.jar;%LIB%/snowball.jar

REM Run BaseX
java -cp "%CP%" %VM% org.basex.BaseX %*
```

Linux/Mac: basex

```
#!/bin/bash

# Path to this script
FILE="${BASH_SOURCE[0]}"
while [ -h "$FILE" ] ; do
  SRC=$(readlink "$FILE")
  FILE="$( cd -P "$(dirname "$FILE")" && \
    cd -P "$(dirname "$SRC")" && pwd )/$(basename "$SRC")"
done
BX="$( cd -P "$(dirname "$FILE")/.." && pwd )"

java -cp "$BX" org.basex.BaseX $*
```

```

# Core and library classes
CP="$BX/BaseX.jar"
CP="$CP$(for JAR in "$BX"/lib/*.jar; do echo -n ":$JAR"; done)"

# Options for virtual machine
VM=-Xmx512m

general_args=( )
vm_args=( )
while (( $# )) ; do
    if [[ $1 = "-X" ]] ; then
        vm_args+=( "$2" )
        shift 2
    else
        general_args+=( "$1" )
        shift
    fi
done

# Run code
java -cp "$CP" $VM "${vm_args[@]}" org.baseX.BaseX "${general_args[@]}"

```

BaseX HTTP Server

The scripts for starting the HTTP server, which gives access to the REST, RESTXQ and WebDAV services, can be found below.

Note: In [Version 7.2](#), the BaseXHTTP start class will be moved from the org.baseX.api package to org.baseX.

Windows: basexhttp.bat

```

@echo off
setlocal

REM Path to this script
set PWD=%~dp0

REM Paths to distributed JAR archives or class files
set BASEX=%PWD%/baseX.jar
set BASEXAPI=%PWD%/baseX-api.jar

REM Classpath
set LIB=%PWD%/../lib
set
CP=%BASEX%;%BASEXAPI%;%LIB%/commons-beanutils-1.8.2.jar;%LIB%/commons-codec-1.4.jar;%LIB%

REM Options for virtual machine
set VM=-Xmx512m

```

```
REM Run code
java -cp "%CP%; ." %VM% org.basex.api.BaseXHTTP %*
```

Linux/Mac: basexhttp

```
#!/bin/bash

# Path to this script
FILE="${BASH_SOURCE[0]}"
while [ -h "$FILE" ] ; do
    SRC="$(readlink "$FILE")"
    FILE="$( cd -P "$(dirname "$FILE")" && \
        cd -P "$(dirname "$SRC")" && pwd )/$(basename "$SRC")"
done
BX="$( cd -P "$(dirname "$FILE")/.." && pwd )"

# API, core, and library classes
CP="$BX/BaseX.jar$(printf ":%s" "$BX/BaseX.jar" "$BX/lib/*.jar")"

# Options for virtual machine
VM=-Xmx512m

general_args=( )
vm_args=( )
while (( $# )) ; do
    if [[ $1 = "-X" ]] ; then
        vm_args+=( "$2" )
        shift 2
    else
        general_args+=( "$1" )
        shift
    fi
done

# Run code
java -cp "$CP" $VM "${vm_args[@]}" org.basex.BaseXHTTP "${general_args[@]}"
```

Changelog

Version 7.2

- The `BaseXHTTP` start class will be moved from `org.basex.api` to `org.basex`.

Version 7.0

- The `basexjaxrx` scripts have been replaced with the `basexhttp` scripts.

References

- [1] <https://github.com/BaseXdb/basex/tree/master/etc>

User Interfaces

Graphical User Interface

This page is part of the Getting Started Section. The BaseX homepage gives you a visual impression ^[1] of the graphical user interface (GUI) of BaseX, and the introductory video ^[2] presents some of the interactive features that the BaseX GUI provides. This small tutorial demonstrates how easy it is to perform simple XPath queries.

Startup

First of all, please launch a GUI instance of BaseX. Depending on your operating system, double click on the **BaseX GUI** start icon or run the `basexgui` script. Beside that, some more startup options are available.

Create Database

Select *Database* → *New* and browse to an XML document of your choice. As an example, you can start with the `factbook.xml` document, which contains statistical information on the worlds' countries. It is included in our official releases and can also be downloaded ^[3] (1.3 MB).

Next, choose the *OK* button, and BaseX will create a database that you can visually explore.

Realtime Options

Via the *Options* menu, you can change the way how queries are executed and visualized:

- **Realtime Execution:** If realtime execution is enabled, your searches and queries will be executed with each key click and the results will be instantly shown.
- **Realtime Filtering:** If enabled, all visualizations will be limited to the actual results in realtime. If this feature is disabled, the query results are highlighted in the visualizations and can be explicitly filtered with the 'Filter' button.

Querying

Keyword Search

The Keyword Search can be executed in the **Search** mode in the combo box of the main window. This options allows for a simple, keyword-based search in the opened database.

The following syntax is supported:

Query	Description
world	Find tags and texts containing world
=world	Find exact matching text nodes
~world	Find text nodes similar to world
@world	Find attributes and attribute values
@=world	Find exact attribute values
"united world"	Find tags and texts containing the phrase "united world"

XPath/XQuery

Apart from the basic search facilities, BaseX offers far more sophisticated processing options to query your documents. Below are some examples you might give a try. This guide is far from being a comprehensive XQuery reference, but might point you in the right direction.

To execute the following queries, enter them in the XQuery Panel and press ENTER or click on the START button.

XPath provides an easy facility to query your documents in a navigational manner. It is the basic tool of all node-related operations that you encounter when using XQuery. We will start with a trivial example and extend it to our needs.

Example: Find Countries

```
//country
```

tells BaseX to look for all `country` elements in the document. The query is introduced by two slashes `//`, which trigger the traversal of all document nodes. The queries `//country` and `/descendant::country` will return the same results.

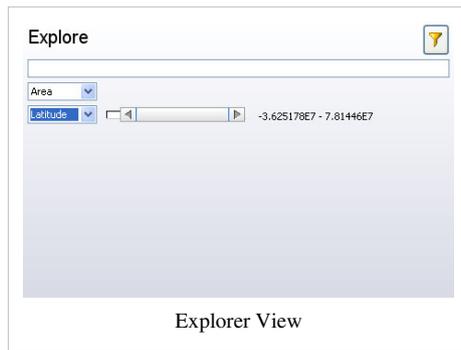
Example: Find Cities in Switzerland

The following query uses a **predicate** `[...]` to filter all `country` nodes which have a `name` child, the string value of which is "Switzerland":

```
//country[name = "Switzerland"]
```

To return all cities of the resulting element node, the query can be extended by a trailing `//city` path:

```
//country[name = "Switzerland"]//city
```

Explorer

With this visualization you can explore the contents of your database via drop-down menus, search fields and double sliders.



Info

This view is helpful for analyzing the query plans of your XQuery expressions. It also displays information on the compilation and evaluation of queries.

What's Next?

Various tutorials on XPath are available in the internet. We invite you to e.g. have a look at the XQuery Tutorial at W3Schools ^[5].

References

- [1] <http://basex.org/products/gui>
- [2] <http://www.youtube.com/watch?v=xILHKGPGaJ4&hd=1>
- [3] <http://files.basex.org/xml/factbook.xml>
- [4] <http://en.wikipedia.org/wiki/Treemap>
- [5] <http://www.w3schools.com/xquery/>

Shortcuts

This page is part of the Getting Started Section. It gives you an overview of the keyboard shortcuts available in the GUI of BaseX.

Global Shortcuts

The following shortcuts are available from most GUI components:

Description	Win/Linux	Mac
Jump to input bar	Ctrl L F6	⌘ L ⌘ F6
Jump to next/previous panel	Ctrl (Shift) Tab	Ctrl (Shift) Tab
Increase/Decrease font size	Ctrl +/-	⌘ +/-
Reset font size	Ctrl 0	⌘ 0

Description	Win/Linux	Mac
Browse back/forward	Alt ←/→ Backspace	⌘ ←/→
Browse one level up	Alt ↑	⌘ ↑
Browse to the root node	Alt Home	⌘ Home

Editor Shortcuts

The text editor can be used to create, edit and save XQuery expressions, XML documents and any other text-based files. It provides numerous shortcuts, which are listed below.

Editing

Description	Win/Linux	Mac
Execute query	Ctrl Enter	⌘ Enter
(Un)comment selection/line	Ctrl K	⌘ K
Undo recent changes	Ctrl Z	⌘ Z
Redo recent changes	Ctrl Y	⌘ Shift Z
Cut selection	Ctrl X Ctrl Delete	⌘ X
Copy selection to clipboard	Ctrl C Ctrl Insert	⌘ C
Paste from clipboard	Ctrl V Shift Insert	⌘ V
Select All	Ctrl A	⌘ A
Delete character left of cursor	Backspace	Backspace
Delete character right of cursor	Delete	Delete (fn Backspace)
Delete word left of cursor	Ctrl Backspace	Alt Backspace
Delete word right of cursor	Ctrl Delete	Alt Delete

Delete text left of cursor	Ctrl Shift Backspace	⌫ Backspace
Delete text right of cursor	Ctrl Shift Delete	⌫ Delete

Find

Description	Win/Linux	Mac
Find text	Ctrl F	⌘ F
Find next instance of text	Ctrl G F3	⌘ G ⌘ F3
Find previous instance of text	Ctrl Shift G Shift F3	⌘ Shift G ⌘ Shift F3

Navigation

Description	Win/Linux	Mac
Move one character to the left/right	←/→	←/→
Move one word to the left/right	Ctrl ←/→	Alt ←/→
Move to beginning/end of line	Home/End	⌘ ←/→
Move one line up/down	↑/↓	↑/↓
Move one screen-full up/down	Page ↑/↓	Page ↑/↓ (fn ↑/↓)
Move to top/bottom	Ctrl Home/End	⌘ ↶/↷ (⌘ ↑/↓)
Scroll one line up/down	Ctrl ↑/↓	Alt ↑/↓

Menu Shortcuts

The following commands and options are also linked from the main menu:

Database

Description	Win/Linux	Mac
Create new database	Ctrl N	⌘ N
Open/manage existing databases	Ctrl O	⌘ O
View/edit database properties	Ctrl D	⌘ D
Close opened database	Ctrl W	⌘ W
Exit application	Ctrl Q	⌘ Q

Query

Description	Win/Linux	Mac
Toggle query/text editor	Ctrl E	⌘ E
Toggle query info view	Ctrl I	⌘ I
Create new text file	Ctrl Shift N	⌘ Shift N
Open existing text file	Ctrl R	⌘ R
Save text file	Ctrl S	⌘ S
Save copy of text file	Ctrl Shift S	⌘ Shift S
Close opened text file	Ctrl Shift W	⌘ Shift W

Help

Description	Win/Linux	Mac
Show Help	F1	F1

View

Description	Win/Linux	Mac
Toggle text view	Ctrl 1	⌘ 1
Toggle map view	Ctrl 2	⌘ 2
Toggle tree view	Ctrl 3	⌘ 3
Toggle folder view	Ctrl 4	⌘ 4
Toggle plot view	Ctrl 5	⌘ 5
Toggle table view	Ctrl 5	⌘ 5
Toggle explorer view	Ctrl 7	⌘ 7

Nodes

Description	Win/Linux	Mac
Copy path to current node	Ctrl Shift C	⌘ Shift C

Options

Description	Win/Linux	Mac
Open preference dialog	Ctrl P	⌘ P

Database Server

This step by step tutorial is part of the Getting Started Guide. It shows you how to run BaseX in client-server mode from a terminal. You can copy and paste all commands to get them running on your machine. After you finished this tutorial, you will be familiar with the basic administration of BaseX. Visit the commands section for a complete list of database commands.

Startup

First of all, please launch a **Server** and **Client** instance of BaseX: double click on the **BaseX Server/Client** icons, or run the `basexserver` and `basexclient` scripts. Follow this link for more information (or check out the additional command-line options).

Create a database

- To create a database you need an XML document, e.g. `factbook.xml` ^[3].
- Save this document to the directory you are working in.
- In the client terminal, type in:

```
> Commands#CREATE DATABASECREATE DB factbook factbook.xml
```

factbook is the name of the database

factbook.xml is the xml file, which is used to create the database

If everything works you see the following lines:

```
Database 'factbook' created in 950.83 ms.
```

Where is the database stored?

By default, databases are stored in the `BaseXData` directory, which is located in your home folder. Depending on your Configuration, the location of your home folder varies. For example, on a Mac it's `/Users/John`, if your name is John. If you have used the Windows Installer, the directory will be named `data`, and reside in the application directory.

Execute a query

The `XQUERY` command lets you run a query.

- For example, this query returns all country nodes in the currently opened database.

```
> Commands#XQUERYXQUERY //country
```

- You can also run queries in files:

```
> Commands#RUNRUN /Users/John/query.xq
```

Create a new database

Now we will create another database from the `xmark.xml` ^[1] document.

- Create the new database, named 'xmark'.
> `Commands#CREATE DATABASECREATE DB xmark xmark.xml`
- Set the new database xmark as the context:
> `Commands#OPENOPEN xmark`
- Now you can easily execute queries on your new database:
> `Commands#XQUERYXQUERY //people/person/name`

Switch the database

- You can explicitly query the factbook database with the `doc(...)` function, no matter what the current context is.
> `Commands#XQUERYXQUERY doc("factbook")//country`
- Otherwise, to set factbook as the current context, execute the following:
> `Commands#OPENOPEN factbook`
- To list the current context, type:
> `Commands#SHOWSHOW DATABASES`

That yields the following lines:

```
1 opened database(s) :
- factbook (1x)
```

Close or delete a database

- To close the current context database, please type:
> `Commands#CLOSECLOSE`
- Use the `DROP` command to delete the xmark database:
> `Commands#DROP DATABASEDROP DB xmark`

Create a collection

What is a collection? With BaseX you can group documents into one logical collection. A collection is a database that contains two or more documents. Collections accept any type of XML documents, regardless of their structure.

Let's add the `xmark.xml` document to the factbook database to create a collection. The name of the original factbook database remains.

- First make sure factbook is opened:
> `Commands#OPENOPEN factbook`
- Now add the `xmark.xml` document:
> `Commands#ADDADD xmark.xml`

Delete a document

- Deleting a document from a collection is easy:
> `Commands#DELETEDELETE xmark.xml`

Make sure that the collection, which contains the **xmark.xml** document, is opened.

Delete a collection

Deleting a collection is the same as deleting a database.

- To delete the collection factbook, type:
> `Commands#DROP DATABASEDROP DB factbook`

Get server information

Several commands help to explore the state of a server. For a complete list, please visit the Commands Section.

- To see all databases on the server, type:
> `Commands#LISTLIST`
- To see which database is currently opened:
> `Commands#SHOW DATABASESSHOW DATABASES`
- To see the general information of the opened database, type:
> `Commands#INFOINFO`
- To see the users in BaseX, type:
> `Commands#SHOW USERSSHOW USERS`

Backup and restore

- To backup your database, type:
> `CREATE BACKUP factbook`
- To restore your database, type:
> `RESTORE factbook`

Where is the backup-file stored?

The backup-file is stored in the database directory. The file is named `factbook-timestamp.zip` (`db_name-timestamp.zip`). To restore the database the file with the newest timestamp is taken.

References

- [1] <http://files.basex.org/xml/xmark.xml>

Standalone Mode

This page is part of the Getting Started Section. BaseX offers a standalone (embedded) console mode from which all database commands can be executed. The article on the Database Server provides numerous examples for running commands in the console mode.

Startup

First of all, please launch a **standalone** version of BaseX: double click on the **BaseX** icon, or run the `basex` script. Follow this link for more information (or check out the additional command-line options).

Working with the BaseX Console

After the BaseX Console has been started, the `HELP` command can be used to list all database commands. Multiple commands can be separated by semicolons.

To evaluate commands without entering the console mode, you can use the `-c` option on the command line:

```
basex -Vc "CREATE DB input <example/>; XQUERY /"
```

```
Database 'input' created in 124.95 ms.
```

```
<example/>
```

```
Query: /
```

```
Compiling:
```

```
Result: root()
```

```
Parsing: 0.42 ms
```

```
Compiling: 9.3 ms
```

```
Evaluating: 0.35 ms
```

```
Printing: 5.53 ms
```

```
Total Time: 15.62 ms
```

```
Hit(s): 1 Item
```

```
Updated: 0 Items
```

```
Printed: 10 Bytes
```

```
Query executed in 15.62 ms.
```

All available command-line options can be found [here](#).

Web Application

BaseX provides access to stored database resources and to the XQuery engine via REST, RESTXQ and WebDAV services. This article describes different ways of deploying and configuring these services. The services can be deployed in 3 different ways:

- as web servlets in a J2EE servlet container,
- for development purposes, using maven, and
- as a standalone application.

Servlet Container

In order to deploy BaseX HTTP Services in a servlet container, you need to download the WAR distribution of BaseX from the download site ^[1]. The WAR file can then be deployed following the instructions of the corresponding servlet container (jetty ^[1], tomcat ^[2]).

Servlet Container Configuration

Configuring port, context path, etc. can be done following the corresponding instructions of the used servlet container. This is needed if you want to replace the default URL path (e.g. `http://localhost:8080/rest`) with a custom one (e.g. `http://localhost:8080/BaseX711/rest`).

Database Access

There are two modes in which the BaseX web application can access databases:

- **local mode:** the web application will access and store data on the local file system.
- **client mode:** the application communicates with a separately started database server.

The operation mode can be configured as explained further below.

Note: if the web application stores data in the local file system of the servlet container, then by default it will use a separate directory as home directory. This is because, in many production environments, the servlet container runs as a dedicated user without a `$HOME` directory. However BaseX needs to store somewhere its settings and data. This is why, even when running a servlet container with a normal user (i.e. one which has a `$HOME` directory), the web application will not use by the default the settings stored in `$HOME/.basex` and will have a different set (or empty set) of databases than the one in the case of the standalone HTTP server. However, this behavior can be changed by manually setting the BaseX home directory to `$HOME`, in which case the web application will use the same databases as in the case of the standalone server.

Configuring Available Services

To enable or disable one of the provided services, the corresponding servlet entry in the `web.xml` file needs to be removed/commented. The default URL paths are listed in the following table:

Service	URL	usage
REST	<code>http://[host]:[port]/[servlet_context_path]/rest</code>	access XML database and its resources, see REST
RESTXQ	<code>{{Mono}http://[host]:[port]/[servlet_context_path]/restxq</code>	create XQuery Web Services, see RESTXQ
WebDAV	<code>http://[host]:[port]/[servlet_context_path]/webdav</code> or <code>webdav://[host]:[port]/[servlet_context_path]/webdav</code> (depending on client)	access databases via the filesystem, see WebDAV
Web Server	<code>http://[host]:[port]/[servlet_context_path]</code>	access your standard web files (e.g. HTML, JavaScript or CSS).

Configuring Database Access

The following options are available to configure the way the databases are accessed. These can be set in the `web.xml` file or as Java system variables (e.g. using `-Dorg.basex.mode=client` from the command line).

Option	Default	Description
<code>org.basex.mode</code>	<code>local</code>	Operation mode. By default, the value is "local": the servlets will access and store data locally using <code>org.basex.path</code> . If "client" is specified, all operations are performed on a remote server instance (the login data of which is stored in the <code>.basex</code> configuration file).
<code>org.basex.user</code>	-	Database user. By default, no value is specified, and the credentials must be specified by the client. Note that it is a clear security risk to store user credentials in the <code>web.xml</code> file. The default user credentials can be found under Startup: BaseX Client.
<code>org.basex.password</code>	-	Database user password.

Additionally, all database main options, which are usually stored in the `.basex` configuration file (DBPATH, HTTPPATH, etc.), can be specified as well by prefixing the key with `org.basex..` If an option references a relative path, it will be prefixed with the servlet root path. Some examples follow:

```
<context-param>
  <param-name>org.basex.dbpath</param-name>
  <param-value>data</param-value>
</context-param>
<context-param>
  <param-name>org.basex.httppath</param-name>
  <param-value>http</param-value>
</context-param>
<context-param>
  <param-name>org.basex.repopath</param-name>
  <param-value>repo</param-value>
</context-param>
<context-param>
  <param-name>org.basex.lang</param-name>
  <param-value>English</param-value>
</context-param>
```

How to set these options in the `web.xml` of the BaseX web application is specific to the servlet container. For example, in Jetty it is done using an overriding `web.xml` ^[3]. Another option is to directly edit the `WEB-INF/web.xml` file in the WAR archive (WAR files are simple ZIP files).

Maven

Checkout the sources of `basex` and `basex-api` as described under Integrate: Check Out. Execute `mvn install` in the `basex` project folder and then `mvn install jetty:run` in the `basex-api` project folder. This will start a Jetty instance in which the servlets will be deployed.

Configuration

The same options as in the case of deployment in a servlet container apply. In this case, however, there is no WAR archive. Instead, Jetty uses the directory `basex-api/src/main/webapp`. For configuring various Jetty runtime options, such as port, context path, etc. refer to the Maven Jetty Plugin ^[4]. These should be entered in the `pom.xml` file of the `basex-api` project.

Standalone Application

Detailed description how to start and setup the standalone BaseX HTTP server can found under Startup: BaseX HTTP Server.

Changelog

Version 7.2

- Web Application concept revised

References

- [1] <http://docs.codehaus.org/display/JETTY/WebAppDeployer>
- [2] <http://tomcat.apache.org/tomcat-6.0-doc/deployer-howto.html>
- [3] <http://docs.codehaus.org/display/JETTY/override+web.xml>
- [4] <http://docs.codehaus.org/display/JETTY/Maven+Jetty+Plugin>

General Info

Databases

This page is part of the Getting Started Section.

In BaseX, a database is a pretty light-weight structure and can be compared with a *collection*. It contains an arbitrary number of **resources**, addressed by their unique database path. Resources can either be **XML documents** or **raw files** (binaries). Some information on binary data can be found on an extra page.

Create Databases

New databases can be created via commands, in the GUI, or with any of our APIs. If some input is specified along with the create operation, it will be added to the database in a bulk operation:

- Console: `CREATE DB db /path/to/resources` will add initial documents to a database
- GUI: Go to *Database* → *New*, press *Browse* to choose an initial file or directory, and press *OK*

Database must follow the valid names constraints. Various parsers can be chosen to influence the database creation, or to convert different formats to XML.

Access Resources

Stored resources and external documents can be accessed in different ways:

XML Documents

Updated with Version 7.2.1:

Various XQuery functions exist to access XML documents in databases and from other locations:

Function	Example	Description
<code>db:open()</code>	<code>db:open("db", "path/to/tocs")</code>	Returns all documents that are found in the database <code>db</code> at the (optional) path <code>path/to/docs</code> .
<code>fn:collection()</code> [1]	<code>collection("db/path/to/docs")</code>	Returns all documents at the location <code>path/to/docs</code> in the database <code>db</code> . If no path is specified after the database, all documents in the database will be returned. If no argument is specified, all documents of the currently opened database will be returned.
<code>fn:doc()</code> [2]	<code>doc("db/path/to/doc.xml")</code>	Returns the document at the location <code>path/to/docs</code> in the database <code>db</code> . An error is raised if the specified addresses does not address exactly one document.

The `fn:document-uri()` and `fn:base-uri()` functions return URIs that can be reused as arguments for the `fn:doc()` and `fn:collection()` functions. As a result of this, as an example, the following query will always return `true`:

```
every $c in collection('anyDB')
satisfies doc-available(document-uri($c))
```

If the argument of `fn:doc()` or `fn:collection()` does not start with a valid database name, or if the addressed database does not exist, the string is interpreted as URI reference, and the documents found at this location will be returned. Examples:

- `doc("http://web.de")`: retrieves the addressed URI and returns it as a main-memory document node.
- `collection("/path/to/docs")`: returns a main-memory collection with all XML documents found at the addressed file path.

Raw Files

- XQuery: `db:retrieve("dbname", "path/to/docs")` returns raw files in their Base64 representation. By choosing `"method=raw"` as Serialization Option, the data is returned in its original byte representation:

```
declare option output:method "raw";
db:retrieve('multimedia', 'sample.avi')
```

- Commands: `RETRIEVE` returns raw files without modifications.

HTTP Services

- With REST and WebDAV, all database resources can be requested in a uniform way, no matter if they are well-formed XML documents or binary files.

Update Resources

Once you have created a database, additional commands exist to modify its contents:

- XML documents can be added with the `ADD` command.
- Raw files are added with `STORE`.
- Resource can be replaced with other ones with the `REPLACE` command.
- Resources can be deleted via `DELETE`.

The `AUTOFLUSH` option can be turned off before *bulk operations* (i.e. before a large number of new resources is added to the database).

The following commands create an empty database, add two resources, explicitly flush data structures to disk, and finally delete all inserted data:

```
CREATE DB example
SET AUTOFLUSH false
ADD example.xml
ADD ...
STORE TO images/ 123.jpg
FLUSH
DELETE /
```

You may as well use the BaseX-specific XQuery Database Functions to add, replace and delete XML documents:

```
let $root := "/path/to/xml/documents/"
for $file in file:list($root)
return db:add("database", $root || $file)
```

Last but not least, XML documents can also be added via the GUI and the *Database* menu.

Export Data

All resources stored in a database can be *exported*, i.e., written back to disk. This can be done in several ways:

- **Commands:** `EXPORT` writes all resources to the specified target directory
- **GUI:** Go to *Database* → *Export*, choose the target directory and press *OK*
- **WebDAV:** Locate the database directory (or a sub-directory of it) and copy all contents to another location

Changelog

Version 7.2.1

- Updated: `fn:document-uri()` and `fn:base-uri()` now return strings that can be reused with `fn:doc()` or `fn:collection()` to reopen the original document.

References

[1] http://www.xqueryfunctions.com/xq/fn_collection.html

[2] http://www.xqueryfunctions.com/xq/fn_doc.html

Binary Data

This page is linked from the Database page.

Since [Version 7.0](#), the BaseX store has been extended to also support binary (*raw*) files. Each database may contain both XML documents and binary files. Both XML and binary data is handled in a uniform way: a unique database path serves as key, and the contents can be retrieved via database commands, XQuery, or the various APIs.

Storage

XML documents are stored in a proprietary format to speed up XPath axis traversals and update operations, and raw data is stored in its original format in a dedicated sub-directory (called "*raw*"). Several reasons exist why we did not extend our existing storage to binary data:

- **Good Performance:** the file system generally performs very well when it comes to the retrieval and update of binary files.
- **Key/Value Stores:** we do not want to compete with existing key/value database solutions. Again, this is not what we are after.
- **Our Focus:** our main focus is the efficient storage of hierarchical data structures and file formats such as XML or (more and more) JSON. The efficient storage of arbitrary binary resources would introduce many new challenges that would distract us from more pressing tasks.

For some use cases, the chosen database design may bring along certain limitations:

- **Performance Limits:** most file system are not capable of handling thousands or millions of binary resources in a single directory in an efficient way. The same problem happens if you have a large number of XML documents that need to be imported in or exported from a BaseX database. The general solution to avoid this bottleneck is to distribute the relevant binaries in additional sub-directories.
- **Keys:** if you want to use arbitrary keys for XML and binary resources, which are not supported by the underlying file system, you may either add an XML document in your database that contains all key/path mappings.

In the latter case, a key/value store might be the better option anyway.

Usage

More information on how to store, retrieve, update and export binary data is found in the general Database documentation.

Parsers

This article is part of the Getting Started Section. It presents different parsers for importing various data source into BaseX databases. For export see [Serialization](#).

XML Parsers

BaseX provides two parsers to import XML data:

- By default, the internal, built-in XML parser is used, which is more fault-tolerant than Java's XML parser. It supports standard HTML entities out-of-the-box, and is faster in most cases. In turn, it does not support all oddities specified by DTDs, and cannot resolve catalogs.
- Java's SAXParser ^[1] can also be selected for parsing XML documents. This parser is stricter than the built-in parser, but it refuses to process some large documents.

GUI

Go to Menu *Database* → *New*, then choose the *Parsing* tab and (de)activate *Use internal XML parser*. The parsing of DTDs can be turned on/off by selecting the checkbox below.

Command Line

To turn the internal XML parser and DTD parsing on/off, modify the `INTPARSE` and `DTD` options:

```
SET INTPARSE true
SET DTD true
```

XQuery

The `db:add()` or `db:replace()` function can be used as well to add new XML documents to the database. The following example query uses the internal parser and adds all files to the database `DB` that are found in the directory `2Bimported`:

```
declare option db:intparse "yes";
for $file in file:list("2Bimported")
return db:add('DB', $file)
```

HTML Parser

With TagSoup ^[2], HTML can be imported in BaseX without any problems. TagSoup ensures that only well-formed HTML arrives at the XML parser (correct opening and closing tags, etc.). Hence, if TagSoup is not available on a system, there will be a lot of cases where importing HTML fails, no matter whether you use the GUI or the standalone mode.

Installation

If the TagSoup classes are accessible via the classpath, or if you run BaseX from the sources and use the Maven build manager, BaseX will automatically use TagSoup to prepare HTML input. TagSoup is also included in the complete BaseX distributions (BaseX.zip, BaseX.exe, etc.) or can be manually downloaded and embedded on the appropriate platforms. Using Debian, TagSoup will be automatically included after it has been installed via:

```
apt-get install libtagsoup-java
```

Options

Introduced with Version 7.2:

TagSoup offers a variety of options to customize the import of HTML. For the complete list please visit the TagSoup ^[2] website. BaseX supports most of these options with a few exceptions:

- **encoding:** BaseX tries to guess the input encoding but this can be overwritten by the user if necessary.
- **files:** Not supported as input documents are piped directly to the XML parser.
- **method:** Set to 'xml' as default. If this is set to 'html' ending tags may be missing for instance.
- **version:** Dismissed, as TagSoup always falls back to 'version 1.0', no matter what the input is.
- **standalone:** Deactivated.
- **pyx, pyxin:** Not supported as the XML parser can't handle this kind of input.
- **output-encoding:** Not supported, BaseX already takes care of that.
- **reuse, help:** Not supported.

GUI

Go to Menu *Database* → *New* and select "HTML" in the input format combo box. There's an info in the "Parsing" tab about whether TagSoup is available or not. The same applies to the "Resources" tab in the "Database Properties" dialog.

These two dialogs come with an input field 'Parameters' where TagSoup options can be entered.

Command Line

Turn on the HTML Parser before parsing documents, and set a file filter:

```
SET PARSER html
SET HTMLOPT method=xml,nons=true,ncdata=true,nodefaults=true,nobogons=true,nocolons=true,ignorable=true
SET CREATEFILTER *.html
```

XQuery

```
declare option db:parser "html";
declare option db:htmlopt "html=false";
doc("index.html")
```

JSON Parser

BaseX can also import JSON documents:

GUI

Go to Menu *Database* → *New* and select "JSON" in the input format combo box. You can set the following options for parsing JSON documents in the "Parsing" tab:

- **Encoding:** Choose the appropriate encoding of the JSON file.
- **JsonML:** Activate this option if the incoming file is a JsonML file.

Command Line

Turn on the JSON Parser before parsing documents, and set some optional, parser-specific options and a file filter:

```
SET PARSER json
SET PARSEROPT encoding=utf-8, jsonml=true
SET CREATEFILTER *.json
```

CSV Parser

BaseX can be used to import CSV documents. Different alternatives how to proceed are shown in the following:

GUI

Go to Menu *Database* → *New* and select "CSV" in the input format combo box. You can set the following options for parsing CSV documents in the "Parsing" tab:

- **Encoding:** Choose the appropriate encoding of the CSV file.
- **Separator:** Choose the column separator of the CSV file (possible: comma, semicolon, tab).
- **XML format:** Choose the XML format (possible: verbose, simple).
- **Header:** Activate this option if the incoming CSV files have a header line.

Command Line

Turn on the CSV Parser before parsing documents, and set some optional, parser-specific options and a file filter:

```
SET PARSER csv
SET PARSEROPT encoding=utf-8, lines=true, format=verbose, header=false, separator=comma
SET CREATEFILTER *.csv
```

XQuery

The CSV parser can also be specified in the prolog of an XQuery expression. The `db:add()` or `db:replace()` function can be used to add the specified source files into the database. The following example query adds all CSV files to the database `DB` that are found in the directory `2Bimported`, and interprets the first lines as column headers:

```
declare option db:parser "csv";
declare option db:parseropt "header=yes";
```

```
for $file in file:list("2Bimported", false(), "*.csv")
return db:add('DB', $file)
```

Text Parser

Plain text can be imported as well:

GUI

Go to Menu *Database* → *New* and select "TEXT" in the input format combobox. You can set the following option for parsing text documents in the "Parsing" tab:

- **Encoding:** Choose the appropriate encoding of the text file.
- **Lines:** Activate this option to create a `<line>...</line>` element for each line of the input text file.

Command Line

Turn on the CSV Parser before parsing documents and set some optional, parser-specific options and a file filter:

```
SET PARSER text
SET PARSEOPT lines=yes
SET CREATEFILTER *
```

XQuery

Again, the text parser can also be specified in the prolog of an XQuery expression, and the `db:add()` or `db:replace()` function can be used to add the specified source files into the database. The following example query adds all text files to the database `DB` that are found in the directory `2Bimported` and its sub-directories:

```
declare option db:parser "text";
for $file in file:list("2Bimported", true(), "*.txt")
return db:add('DB', $file)
```

Changelog

Version 7.2

- Enhanced support for TagSoup options.

References

- [1] <http://download.oracle.com/javase/6/docs/api/javax/xml/parsers/SAXParser.html>
- [2] <http://home.cci1.org/~cowan/XML/tagsoup/>

Commands

This article is linked from the Getting Started Section. It lists all database commands supported by BaseX. Commands can e.g. be run from the Command Line, the Clients, REST, the input field in the GUI, and in numerous other ways. If the GUI is used, all commands that are triggered by the GUI itself will show up in the Info View. The Permission fields indicate which rights are required by a user to perform a command in the client/server architecture. A shortcut exists for some of the command keywords. For example, you may replace the `DATABASE` keyword with `DB`.

Database Operations

CREATE DATABASE

Signature `CREATE DATABASE [name] ([input])`

Permission `CREATE`

Summary Creates the database `[name]` with an optional `[input]`.
The input may either be a reference to a single XML document, a directory, a remote URL, or a string containing XML. `[name]` must be a valid database name.

Errors The command fails if a database with the specified name is currently used by another process, if one of the documents to be added is not well-formed or if it cannot be parsed for some other reason.

Examples

- `CREATE DATABASE input`
creates an empty database `input`.
- `CREATE DATABASE xmark [1]`
creates the database `xmark`, containing a single initial document called `xmark.xml`.
- `CREATE DATABASE coll /path/to/input`
creates the database `coll` with all documents found in the `input` directory.
- `SET INTPARSE false; CREATE DATABASE input input.xml`
creates a database `input` with `input.xml` as initial document, which will be parsed with Java's default XML parser.

OPEN

Signature `OPEN [path]`

Permission `READ`

Summary Opens a database or some of its documents. `[path]` is the name of the database. If the name is further refined by a path, only some of the documents in the database will be opened.

Errors The command fails if the specified database does not exist, is currently being updated by another process, cannot be opened for some other reason.

CHECK

Signature CHECK [input]

Permission READ/CREATE

Summary This command combines OPEN and CREATE DATABASE: if a database with the name [input] exists, it is opened. Otherwise, it creates a new database and stores the specified input as initial content.

Errors The command fails if the addressed database could neither be opened nor created.

CLOSE

Signature CLOSE

Permission READ

Summary Closes the currently opened database.

Errors The command fails if the database files could not be closed for some reason.

EXPORT

Signature EXPORT [path]

Permission CREATE

Summary Exports all documents in the database to the specified [path], using the serializer options specified by the EXPORTER option.

Errors The command fails if no database is opened, if the target path points to a file or is invalid, if the serialization parameters are invalid, or if the documents cannot be serialized for some other reason.

CREATE INDEX

Updated in Version 7.2: PATH removed as argument; path summary is now updated by OPTIMIZE command

Signature CREATE INDEX [TEXT|ATTRIBUTE|FULLTEXT]

Permission WRITE

Summary Creates the specified database index.

Errors The command fails if no database is opened, if the specified index is unknown, or if indexing fails for some other reason.

DROP INDEX

Updated in Version 7.2: PATH removed as argument; path summary is now always available

Signature DROP INDEX [TEXT|ATTRIBUTE|FULLTEXT]

Permission WRITE

Summary Drops the specified database index.

Errors The command fails if no database is opened, if the specified index is unknown, or if it could not be deleted for some other reason.

Administration

ALTER DATABASE

Signature ALTER DATABASE [name] [newname]

Permission CREATE

Summary Renames the database specified by [name] to [newname]. [newname] must be a valid database name.

Errors The command fails if the target database already exists, if the source database does not exist or is currently locked, or if it could not be renamed for some other reason.

Examples

- ALTER DATABASE db tempdb
renames the database db into tempdb.

DROP DATABASE

Signature DROP DATABASE [name]

Permission CREATE

Summary Drops the database with the specified [name]. The Glob Syntax can be used to address more than one database.

Errors The command fails if the specified database does not exist or is currently locked, or if the database could not be deleted for some other reason.

CREATE BACKUP

Signature CREATE BACKUP [name]

Permission CREATE

Summary Creates a zipped backup of the database specified by [name]. The backup file will be suffixed with the current timestamp and stored in the database directory. The Glob Syntax can be used to address more than one database.

Errors The command fails if the specified database does not exist, or if it could not be zipped for some other reason.

Examples

- BACKUP db
creates a zip archive of the database db (e.g. db-2011-04-01-12-27-28.zip) in the database directory.

RESTORE

Signature RESTORE [name]

Permission CREATE

Summary Restores a database with the specified [name]. The name may include the timestamp of the backup file.

Errors The command fails if the specified backup does not exist, if the database to be restored is currently locked, or if it could not be restored for some other reason.

DROP BACKUP

Signature DROP BACKUP [name]

Permission CREATE

Summary Drops all backups of the database with the specified [name]. The Glob Syntax can be used to address more than one database.

Examples

- DROP BACKUP abc*
deletes the backups of all databases starting with the characters abc.

SHOW BACKUPS

Signature SHOW BACKUPS

Permission CREATE

Summary Shows all database backups.

COPY

Signature COPY [name] [newname]

Permission CREATE

Summary Creates a copy of the database specified by [name]. [newname] must be a valid database name.

Errors The command fails if the target database already exists, or if the source database does not exist.

INFO DATABASE

Signature INFO DATABASE

Permission READ

Summary Shows information on the currently opened database.

Errors The command fails if no database is opened.

INFO INDEX

Signature `INFO INDEX ([TEXT|ATTRIBUTE|FULLTEXT|PATH])`

Permission `READ`

Summary Shows information on the existing index structures. The output can be optionally limited to the specified index.

Errors The command fails if no database is opened, or if the specified index is unknown.

INFO STORAGE

Signature `INFO STORAGE [start end] | [query]`

Permission `READ`

Summary Shows the internal main table of the currently opened database. An integer range or a query may be specified as argument.

Errors The command fails if no database is opened, or if one of the specified arguments is invalid.

Querying

LIST

Signature `LIST ([path])`

Permission `NONE`

Summary Lists all available databases, or the documents in a database. `[path]` is the name of the database, optionally followed by a path to the requested documents.

Errors The command fails if the optional database cannot be opened, or if the existing databases cannot be listed for some other reason.

XQUERY

Signature `XQUERY [query]`

Permission *depends on query*

Summary Runs the specified `[query]` and prints the result.

Errors The command fails if the specified query is invalid.

Examples

- `XQUERY 1 to 10`
returns the sequence (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).
- `SET RUNS 10; XQUERY 1 to 10`
runs the query 10 times, returns the result and prints the average execution time.
- `SET XMLPLAN true; XQUERY 1 to 10`
returns the result and prints the query plan as XML.

RETRIEVE

Signature RETRIEVE [path] [input]

Permission *READ*

Summary Retrieves raw data from the specified database [path].

Errors The command fails if no database is opened, if the source path is invalid or if the data cannot not be retrieved for some other reason.

RUN

Signature RUN [file]

Permission *depends on query*

Summary Runs the query contained in [file] and prints the result.

Errors The command fails if the specified file does not exist, or if the retrieved query is invalid.

FIND

Signature FIND [keywords]

Permission *READ*

Summary Builds and runs a query for the specified [keywords]. Keywords can be enclosed in quotes to look for phrases. The following modifiers can be used to further limit search: = looks for exact text nodes
~ looks for approximate hits
@= looks for exact attribute values
@ looks for attributes

Errors The command fails if no database is opened.

CS

Signature CS [query]

Permission *depends on query*

Summary Evaluates the specified [query] and sets the result as new context set.

Errors The command fails if no database is opened, if the specified query is invalid or if it does not return nodes of the currently opened database.

REPO INSTALL

Signature REPO INSTALL [path]

Permission CREATE (before Version 7.2: ADMIN)

Summary Installs the package with path [path].

Errors The command fails in the following cases:

- The package to be installed is not a xar file.
- The package to be installed does not exist or is already installed.
- The package descriptor is with invalid syntax.
- The package to be installed depends on a package which is not installed.
- The package is not supported by the current version of BaseX.
- A component of the package is already installed as part of another package.

REPO LIST

Signature REPO LIST

Permission READ (before Version 7.2: ADMIN)

Summary Lists all installed packages.

REPO DELETE

Signature REPO DELETE [name|dir]

Permission CREATE (before Version 7.2: ADMIN)

Summary Deletes the package with name [name] or with directory [dir].

Errors The command fails if the package to be deleted participates in a dependency.

Updates

ADD

Signature ADD (TO [path]) [input]

Permission WRITE

Summary Adds the files, directory or XML string specified by [input] to the currently opened database at the specified [path]. [input] may either be a single XML document, a directory, a remote URL or a plain XML string. If the path denotes a directory, it needs to be suffixed with a slash (/).

Errors The command fails if no database is opened, if one of the documents to be added is not well-formed, or if it could not be parsed for some other reason.

Examples

- ADD input.xml
adds the file input.xml to the database.
- ADD TO temp/one.xml input.xml
adds input.xml to the database and moves it to temp/one.xml.
- ADD TO target/ xmldir
adds all files from the xmldir directory to the database in the target path.

DELETE

Signature DELETE [path]

Permission WRITE

Summary Deletes all documents from the currently opened database that start with the specified [path].

Errors The command fails if no database is opened.

RENAME

Signature RENAME [path] [newpath]

Permission WRITE

Summary Renames all document paths in the currently opened database that start with the specified [path]. The command may be used to either rename single documents or directories.

Errors The command fails if no database is opened, or if the target path is empty.

Examples

- RENAME one.xml two.xml
renames the document one.xml to two.xml.
- RENAME / TOP
moves all documents to a TOP root directory.

REPLACE

Signature REPLACE [path] [input]

Permission WRITE

Summary Replaces the documents in the currently opened database, addressed by [path], with the file or XML string specified by [input]. The original file name and path is preserved by the operation.

Errors The command fails if no database is opened, if the specified path points to a database directory, or if the input is not found.

Examples

- REPLACE one.xml input.xml
replaces the document one.xml with the contents of the file input.xml.
- REPLACE top.xml <xml/>
replaces the document top.xml with the document <xml/>.

STORE

Signature STORE (TO [path]) [input]

Permission WRITE

Summary Stores raw data to the specified [path]. [input] may either be a file reference, a remote URL, or a plain string. If the path denotes a directory, it needs to be suffixed with a slash (/).

Errors The command fails if no database is opened, if the specified resource is not found, if the target path is invalid or if the data cannot not be written for some other reason.

OPTIMIZE

Signature OPTIMIZE (ALL)

Permission WRITE

Summary Optimizes the index structures, meta data and statistics of the currently opened database. If the ALL flag is specified, the internal database structures are completely rebuilt; this often leads to a reduction of the total database size.

Errors The command fails if no database is opened, or if the currently opened database is a main-memory instance.

FLUSH

Signature FLUSH

Permission WRITE

Summary Explicitly flushes the buffers of the currently opened database to disk. This command is applied if the AUTOFLUSH option has been set to false.

Errors The command fails if no database is opened.

Server Administration

SHOW DATABASES

Signature SHOW DATABASES

Permission ADMIN

Summary Shows all databases that are opened in the current server instance.

SHOW SESSIONS

Signature SHOW SESSIONS

Permission ADMIN

Summary Shows all sessions that are connected to the current server instance.

SHOW USERS

Signature SHOW USERS (ON [database])

Permission ADMIN

Summary Shows all users that are registered in the database. If a [database] is specified, local users are shown.

Errors The command fails if the optional database could not be opened.

KILL

Signature KILL [target]

Permission ADMIN

Summary Kills sessions of a user or an IP:port combination, specified by [target]. The Glob Syntax can be used to address more than one user.

Errors The command fails if a user tried to kill his/her own session.

CREATE EVENT

Signature CREATE EVENT [NAME]

Permission ADMIN

Summary Creates the specified event.

Errors The command fails if event already exists.

SHOW EVENTS

Signature SHOW EVENTS

Permission ADMIN

Summary Shows all events that have been registered in the database.

DROP EVENT

Signature DROP EVENT [NAME]

Permission ADMIN

Summary Drops the specified event.

Errors The command fails if the event doesn't exist.

User Management

CREATE USER

Signature CREATE USER [name] ([password])

Permission ADMIN

Summary Creates a user with the specified [name] and [password]. [name] must be a valid user name. The password must be a valid MD5 hash value. If no password is specified in the console mode, it is requested via standard input.

Errors The command fails if the specified user already exists, or if the password is no valid MD5 hash value.

ALTER USER

Signature ALTER USER [name] ([password])

Permission ADMIN

Summary Alters the [password] of the user specified by [name]. The password must be a valid MD5 hash value. If no password is specified in the console mode, it is requested via standard input.

Errors The command fails if the specified user does not exist, or if the password is no valid MD5 hash value.

DROP USER

Signature DROP USER [name] (ON [database]):

Permission ADMIN

Summary Drops the user with the specified [name]. If a [database] is specified, the user is only dropped locally. The Glob Syntax can be used to address more than one database or user.

Errors The command fails if admin is specified as user name, if the specified user does not exist or is logged in, or if the optional database could not be opened for modification.

GRANT

Signature GRANT [NONE|READ|WRITE|CREATE|ADMIN] (ON [database]) TO [user]

Permission ADMIN

Summary Grants the specified permission to the specified [user]. If a [database] is specified, the permissions are only granted locally. The Glob Syntax can be used to address more than one database or user.

Errors The command fails if admin is specified as user name, if the specified user does not exist, or if the optional database could not be opened for modification.

Examples

- GRANT READ TO JoeWinson
grants READ permission to the user JoeWinson.
- GRANT WRITE ON Wiki TO editor*
grants WRITE permissions on the Wiki database to all users starting with the characters editor*.

PASSWORD

Signature PASSWORD ([password])

Permission NONE

Summary Changes the [password] of the current user. The password must be a valid MD5 hash value. If no password is specified in the console mode, it is requested via standard input.

Errors The command fails if the password is no valid MD5 hash value.

General Commands

GET

Updated in Version 7.2.1: permission changed from READ to NONE

Signature GET [option]

Permission NONE

Summary Returns the current value of the Option specified via [key].

Errors The command fails if the specified option is unknown.

SET

Updated in Version 7.2.1: permission changed from READ to NONE

Signature SET [option] ([value])

Permission NONE

Summary Sets the Option with the specified [key] to a new [value]. If no value is specified, and if the value is boolean, it will be inverted.

Errors The command fails if the specified option is unknown or if the specified value is invalid.

INFO

Signature INFO

Permission READ

Summary Shows global information.

HELP

Signature `HELP ([command])`

Permission `NONE`

Summary If `[command]` is specified, information on the specific command is printed; otherwise, all commands are listed.

Errors The command fails if the specified command is unknown.

EXIT

Signature `EXIT`

Permission `NONE`

Summary Exits the console mode.

Conventions

Glob Syntax

For some commands, the glob syntax can be used to address more than one database or user. Question marks and asterisks can be used to match one or more characters, and commas can be used to separate multiple patterns. Some examples:

- `AB?` addresses all names with the characters `AB` and one more character.
- `*AB` addresses all names ending with the characters `AB`.
- `X*, Y*, Z*` addresses all names starting with the characters `X`, `Y`, or `Z`.

Valid Names

Both database and user names must follow the same naming constraints. Valid names may contain letters, numbers, underscores and dashes. Names must have at least one character; they also should not be longer than 128 characters, although this is not enforced. A regular expression matching valid names is `[-_a-zA-Z0-9]{1,128}`.

Shortcuts

In all commands, the `DATABASE` keyword can be replaced by the shortcut `DB`.

Changelog

Version 7.2.1

- Updated: permissions for `GET` and `SET` changed from `READ` to `NONE`

Version 7.2

- Updated: `CREATE INDEX`, `DROP INDEX` (`PATH` argument removed. Path summary is always available now and updated with `OPTIMIZE`)
- Updated: permissions for `REPO DELETE`, `REPO INSTALL` and `REPO LIST`

Version 7.1

- Updated: `KILL` (killing sessions by specifying `IP:port`)

Version 7.0

- Added: `FLUSH`, `RETRIEVE`, `STORE`
- Updated: `ADD`: simplified arguments

Options

This page is linked from the Getting Started Section.

The options listed on this page can be requested with the `GET` command and changed with the `SET` command. Three data types exist: strings, numbers, and booleans, which can be turned `ON` and `OFF`. If options are internally changed by the GUI of BaseX, they will be listed in the Info View.

Main Options

The main options are only available from the Standalone Mode and Database Server instances of BaseX, i.e., they cannot be changed by database clients. They are stored in the `.basex` config file, which is opened by every new BaseX instance:

DBPATH

Signature `DBPATH [path]`

Default `{home}/BaseXData` or `{home}/data`

Summary Points to the directory in which all databases are located.

Note: this option can only be changed if no database is currently opened. If the option is changed, existing databases will not be moved to the new location.

REPOPATH

Signature REPOPATH [path]

Default {home}/BaseXRepo

Summary Points to the Repository, in which all XQuery modules are located.

Note: if the option is changed, BaseX needs to be restarted in order to activate existing packages and avoid side effects.

DEBUG

Signature DEBUG

Default false

Summary Sends internal debug info to STDERR. This option can be turned on to get additional information for development and debugging purposes.

LANG

Signature LANG [language]

Default English

Summary Specifies the interface language. Currently, seven languages are available: 'English', 'German', 'French', 'Dutch', 'Italian', 'Japanese', and 'Vietnamese'. BaseX needs to be restarted in order to activate the new language.

LANGKEY

Signature LANGKEY

Default false

Summary Prefixes all texts with the internal language keys. This option is helpful if BaseX is translated into another language, and if you want to see where particular texts are displayed. BaseX needs to be restarted in order to activate this option.

Client/Server Architecture

HOST

Signature HOST [host]

Default localhost

Summary This host name is used by the client when connecting to a server.

PORT

Signature PORT [port]

Default 1984

Summary This port is used by the client when connecting to a server.

SERVERPORT

Signature SERVERPORT [port]

Default 1984

Summary This is the port the database server will be listening to.

SERVERHOST

Signature SERVERHOST [host|ip]

Default *empty (wildcard)*

Summary This is the host name or ip address the server is bound to. The server will be open to all clients if the option is set to an empty string.

EVENTPORT

Signature EVENTPORT [port]

Default 1985

Summary This port is used by the client to listen for server events. This port will only be bound if a client attaches itself to a database event.

HTTPPORT

Signature HTTPPORT [port]

Default 8984

Summary This is the port the HTTP Server will be listening to.

STOPPORT

Signature STOPPORT [port]

Default 8985

Summary This is the port on which the HTTP Server can be locally closed.

HTTPPATH

Signature HTTPPATH [path]

Default {home}/BaseXHTTP

Summary Points to the HTTP root directory, in which HTML files may be stored, and query files that will be evaluated by the REST service.

PROXYHOST

Introduced with Version 7.2:

Signature PROXYHOST [host]

Default

Summary This is the host name of a proxy server.

PROXYPORT

Introduced with Version 7.2:

Signature PROXYPORT [port]

Default 80

Summary This is the port number of a proxy server.

NONPROXYHOSTS

Introduced with Version 7.2:

Signature NONPROXYHOSTS [hosts]

Default

Summary This is a list of hosts that should be directly accessed.

PARALLEL

Signature PARALLEL [number]

Default 8

Summary Denotes the maximum allowed `number` of parallel read transactions.

TIMEOUT

Signature TIMEOUT [seconds]

Default 0 (*no timeout*)

Summary Specifies the maximum time a read-only transaction may take. If an operation takes longer than the specified timeout, it will be aborted.

Write operations will not be affected by this timeout, as this would corrupt the integrity of the database.

The timeout is deactivated if the timeout is set to 0. Since [Version 7.2](#), it is ignored for ADMIN operations.

KEEPALIVE

Signature KEEPALIVE [seconds]

Default 0 (*no timeout*)

Summary Specifies the maximum time a client will be remembered by the server. If there has been no interaction with a client for a longer time than specified by this timeout, it will be disconnected. Running operations will not be affected by this option. The keepalive check is deactivated if the value is set to 0.

Create Options

CHOP

Signature CHOP

Default true

Summary Chops all leading and trailing whitespaces from text nodes while building a database, and discards empty text nodes. This option often reduces the database size by up to 50%.

INTPARSE

Signature INTPARSE

Default true

Summary Uses the internal XML parser instead of the standard Java XML parser. The internal parser is faster, more fault tolerant and supports common HTML entities out-of-the-box, but it does not support all features needed for parsing DTDs.

DTD

Signature DTD

Default false

Summary Parses referenced DTDs and resolves XML entities. By default, this option is switched to `false`, as many DTDs are located externally, which may completely block the process of creating new databases. The CATFILE option can be changed to locally resolve DTDs.

CATFILE

Signature CATFILE [path]

Default *empty*

Summary Specifies a catalog file to locally resolve DTDs; see the entry on Catalog Resolvers for more details.

CREATEFILTER

Signature CREATEFILTER [filter]

Default *.xml

Summary File filter in the Glob Syntax, which is applied whenever new databases are created, or resources are added to a database.

ADDARCHIVES

Signature ADDARCHIVES

Default true

Summary If this option is set to `true`, files within archives (ZIP, GZIP, DOCX, etc.) are parsed whenever new database are created or resources are added to a database.

SKIPCORRUPT

Signature SKIPCORRUPT

Default false

Summary Skips corrupt (i.e., not well-formed) files while creating a database or adding new documents. If this option is activated, document updates are slowed down, as all files will be parsed twice. Next, main memory consumption will be higher as parsed files will be cached in main memory.

ADDRAW

Signature ADDRAW

Default false

Summary If this option is activated, and if new resources are added to a database, all files that are not filtered by the CREATEFILTER option will be added as *raw* files (i.e., in their binary representation).

PARSER

Signature `PARSER [type]`

Default `XML`

Summary Defines a parser for importing new files to the database. Currently, 'XML', 'JSON', 'CSV', 'TEXT', 'HTML' are available as parsers. HTML will be parsed as normal XML files if Tagsoup^[2] is not found in the classpath.

PARSEROPT

Signature `PARSEROPT [options]`

Default `empty`

Summary Defines parser-specific options; see Parsers for more information.

HTMLOPT

Introduced with Version 7.2:

Signature `HTMLOPT [options]`

Default `empty`

Summary Allows to specify TagSoup options for HTML parsing; see HTML Parser for more information.

Database Options

MAINMEM

Signature `MAINMEM`

Default `false`

Summary If this option is turned on, new databases will be exclusively created in main memory. Most queries will be evaluated faster in main memory mode, but all data is lost if BaseX is shut down. The value of this option will be assigned once to a new database, and cannot be changed after that.

PATHINDEX

Signature `PATHINDEX`

Default `true`

Summary Creates a path index whenever a new database is created. A path index helps to optimize location paths; see Indexes for more details.

TEXTINDEX

Signature TEXTINDEX

Default true

Summary Creates a text index whenever a new database is created. A text index speeds up queries with equality comparisons on text nodes; see Indexes for more details.

ATTRINDEX

Signature ATTRINDEX

Default true

Summary Creates an attribute index whenever a new database is created. An attribute index speeds up queries with equality comparisons on attribute values; see Indexes for more details.

FTINDEX

Signature FTINDEX

Default false

Summary Creates a full-text index whenever a new database is created. A full-text index speeds up queries with full-text expressions; see Indexes for more details.

MAXLEN

Signature MAXLEN

Default 96

Summary Specifies the maximum length of strings that are to be indexed by the name, path, value, and full-text index structures. The value of this option will be assigned once to a new database, and cannot be changed after that.

MAXCATS

Signature MAXCATS

Default 100

Summary Specifies the maximum number of distinct values (categories) that will be remembered for a particular tag/attribute name or unique path. The value of this option will be assigned once to a new database, and cannot be changed after that.

UPDINDEX

Signature UPDINDEX

Default `false`

Summary If turned on, incremental indexing will be activated: all update operations will also update the value index structures (texts and attribute values). The value of this option will be assigned once to a new database, and cannot be changed after that. The advantage of incremental indexes is that the value index structures will always be up-to-date. The downside is that updates will take a little bit longer.

WRITEBACK

Signature WRITEBACK

Default `false`

Summary Updates on XML nodes are written back to the input files. Note that no backups of your original files will be created if this option is turned on.

AUTOFLUSH

Signature AUTOFLUSH

Default `true`

Summary Flushes database buffers to disk after each update. If this option is set to `false`, bulk operations (multiple single updates) will be evaluated faster. As a drawback, the chance of data loss increases if the database is not explicitly flushed via the FLUSH command.

MAXSTAT

Signature MAXSTAT [num]

Default 30

Summary Specifies the maximum number of index occurrences printed by the `INFO INDEX` command.

Full-Text Options

WILDCARDS

Signature WILDCARDS

Default `false`

Summary If a new full-text index is created, it will be particularly optimized for wildcard expressions. See the page on Full-Texts for more information on XQuery Full Text.

STEMMING

Signature STEMMING

Default false

Summary A new full-text index will stem all tokens and speed up queries on stemmed tokens. The same stemming normalization will be applied to all query tokens that are checked against tokens in this index.

CASESENS

Signature CASESENS

Default false

Summary A new full-text index will preserve the case of all tokens. The same case normalization will be applied to all query tokens that are checked against tokens in this index.

DIACRITICS

Signature DIACRITICS

Default false

Summary A new full-text index will preserve the diacritics of all tokens. The same diacritics normalization will be applied to all query tokens that are checked against tokens in this index.

LANGUAGE

Signature LANGUAGE [lang]

Default en

Summary A new full-text index will use the given language to normalize all tokens. This option is mainly important if tokens are to be stemmed, or if the tokenization of a language differs from Western languages.

SCORING

Signature SCORING [mode]

Default 0

Summary A new full-text index will pre-calculate full-text scores. This option enables TF/IDF-based scores in full-text queries and increases main memory consumption while indexing. See the Scoring for more information on different scoring types.

STOPWORDS

Signature STOPWORDS [path]

Default *empty*

Summary A new full-text index will drop tokens that are listed in the specified stopword list. A stopword list may decrease the size of the full text index. A standard stopword list for English texts is provided in the directory `etc/stopwords.txt` in the official releases.

LSERROR

Signature LSERROR [error]

Default 0

Summary This option specifies the maximum Levenshtein error for the BaseX-specific fuzzy match option. See the page on Full-Texts for more information on fuzzy querying.

Query Options

QUERYINFO

Signature QUERYINFO

Default *false*

Summary Prints more information on internal query rewritings, optimizations, and performance. The query info will always be printed in the Info View in the GUI, or it can be activated with the `-V` flag on command line.

XQUERY3

Signature XQUERY3

Default *true*

Summary Enables all XQuery 3.0 features supported by BaseX. If this option is set to *false*, the XQuery parser will only accept expressions of the XQuery 1.0 specification.

SERIALIZE

Signature SERIALIZE

Default *true*

Summary Results of XQuery expressions will be serialized if this option is turned on. For debugging purposes and performance measurements, this option can be set to *false*.

BINDINGS

Signature BINDINGS [vars]

Default *empty*

Summary Contains external variables to be bound to a query. Variable names and values are separated by equality signs, and multiple variables are delimited by commas. Variables may optionally be introduced with a leading dollar sign. Commas that occur in the value itself are encoded by duplication. If a variable uses a namespace different to the default namespace, it can be specified with the Clark Notation ^[1] or Expanded QName Notation ^[2].

Examples `$a=1, $b=2` binds the values 1 and 2 to the variables \$a and \$b
`a=1, , 2` binds the value 1, 2 to the variable \$a
`{URI}a=x` or `'URI':a=x` binds the value x to the variable \$a with the namespace URI.

SERIALIZER

Signature SERIALIZER [params]

Default *empty*

Summary Contains parameters for serializing queries; see *Serialization* for more details. Keys and values are separated by equality signs, and multiple parameters are delimited by commas.

Example `encoding=US-ASCII,omit-xml-declaration=no` : sets the encoding to US-ASCII and prints the XML declaration.

EXPORTER

Signature EXPORTER [params]

Default *empty*

Summary Contains parameters for exporting all resources of a database; see *Serialization* for more details. Keys and values are separated by equality signs, multiple parameters are delimited by commas.

QUERYPATH

Signature QUERYPATH [path]

Default *empty*

Summary Contains the path (*base URI*) to the executed query (default: *empty*). This directory will be used to resolve relative paths to documents, query modules, and other resources addressed in a query.

CACHEQUERY

Signature CACHEQUERY

Default false

Summary Caches the query results before returning them to the client. This option may be set to `true` if the whole result is needed for further operations (such as is e.g. the case in the GUI of BaseX).

FORCECREATE

Signature FORCECREATE

Default false

Summary By activating this option, the XQuery `doc()` and `collection()` functions will create database instances for the addressed input files.

RUNS

Signature RUNS [num]

Default 1

Summary Specify number of runs a query is executed. Results are printed a single time and evaluation times are averages of all runs.

Serialization Options

XMLPLAN

Signature XMLPLAN

Default false

Summary Prints the execution plan of an XQuery expression in its XML representation.

COMPPLAN

Signature COMPPLAN

Default true

Summary Creates the query plan before or after the compilation step. Query plans might change due to optimizations.

DOTPLAN

Signature DOTPLAN

Default false

Summary Visualizes the execution plan of an XQuery expression with `dotty`^[1] and saves its dot file in the query directory.

DOTCOMPACT

Signature DOTCOMPACT

Default false

Summary Chooses a compact dot representation.

DOTDISPLAY

Signature DOTDISPLAY

Default true

Summary Visualizes the dot representation after the query execution.

DOTTY

Signature DOTTY [path]

Default dotty

Summary Location of the `dotty` executable.

Changelog

Version 7.2

- Added: PROXYHOST, PROXYPORT, NONPROXYHOSTS, HTMLOPT
- Updated: TIMEOUT: ignore timeout for admin users

Version 7.1

- Added: ADDDRAW, MAXLEN, MAXCATS, UPDINDEX
- Updated: BINDINGS

Version 7.0

- Added: SERVERHOST, KEEPALIVE, AUTOFLUSH, QUERYPATH

References

[1] <http://www.graphviz.org>

Integration

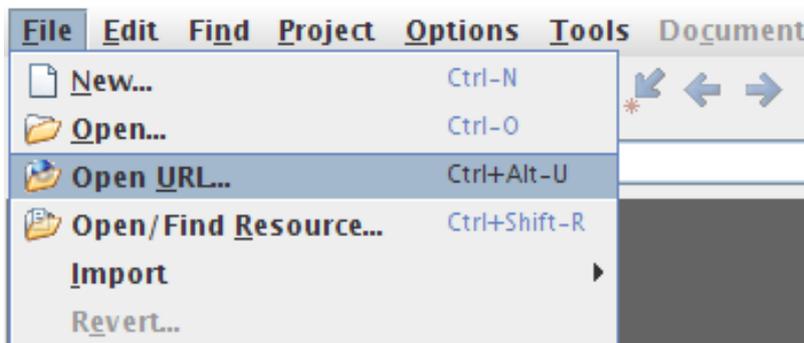
Integrating oXygen

This tutorial is part of the Getting Started Section. It describes how to access BaseX from the oXygen XML Editor ^[1]. Currently, there are two variants how to use BaseX in oXygen:

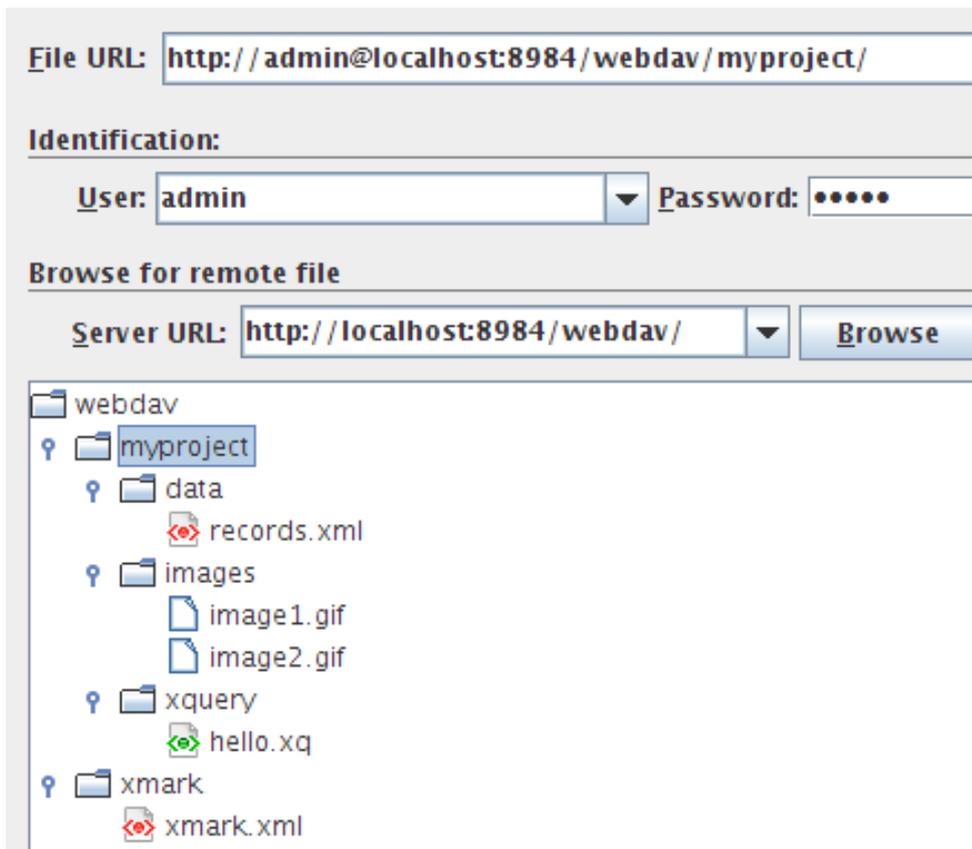
- Resources in databases can be opened and modified.
- XPath/XQuery expressions can be run by the query processor of BaseX.

Access Database Resources

- Start the WebDAV service first, which will allow you to access all resources via the client/server architecture.
- Choose *File* → *Open URL...*



- Enter the corresponding user name and password (if needed), the URL of the BaseX WebDAV Server, and then click "Browse".



Perform Queries

Preparations

1. Download one of the complete BaseX distributions ^[1] (ZIP, EXE)
2. Start oXygen and go to *Options* → *Preferences* → *Data Sources*.
3. Start a BaseX Server instance

Data Source

1. Add a new Data Source with the *New* button.
2. Enter "BaseX" as name and choose *XQuery API for Java (XQJ)* as type.
3. Add the following JAR files above with the *Add* Button: `xqj-api-1.0.jar`, `xqj2-0.0.1.jar` and `basex-xqj-0.9.2.jar` (the version names of the JAR file may differ).
4. Now press *OK*, and your Data Source is ready for use.

Connection

1. Now press *New* in the Connection Panel below.
2. Enter Name "BaseX" and select "BaseX" in the Data Source box.
3. Now press *OK*, and your connection is ready.

Usage

The query execution works as follows:

1. Configure a new transformation scenario in *Window* → *Show View* → *Transformation Scenarios*.
2. Choose the *XQuery Transformation* tree entry.
3. Press the plus sign to add a new scenario.
4. Enter a Name and an optional XML and XQuery URL (e.g. your query document/file).
5. Choose "BaseX" as Transformer from the combo box.
6. Press *OK*, and your scenario is ready. Now you can start the transformation, e.g. by clicking on the red *Play* button.
7. The results should immediately occur in the result panel.

References

[1] <http://www.oxygenxml.com>

Integrating Eclipse

This tutorial is part of the Getting Started Section.

It demonstrates how to perform XPath/XQuery expressions and access databases from the Eclipse IDE ^[1].

Required Software

1. BaseX Java Archive: <http://basex.org/download/>
2. Eclipse 3.6 or higher: <http://www.eclipse.org>
3. XQuery Development Tools Plugin: <http://www.xqdt.org/>
Update Site: <http://download.eclipse.org/webtools/incubator/repository/xquery/milestones/>

Setting up

You can set up the XQuery interpreter as standalone or client version.

Setting up as Standalone

1. Start Eclipse and go to *Preferences* → *XQuery* → *Interpreters*.
 2. Add a new Interpreter with the "Add" button.
 3. Enter "BaseX" as name and choose "Java XQuery Engine" as Interpreter type.
 4. Point "Interpreter JAR/WAR" to the BaseX JAR archive
 5. Choose "org.basex.BaseX" as "Main class"
-

Setting up as Client

1. Start Eclipse and go to *Preferences* → *XQuery* → *Interpreters*.
2. Add a new Interpreter with the "Add" button.
3. Enter "BaseX" as name and choose "Java XQuery Engine" as Interpreter type.
4. Point "Interpreter JAR/WAR" to the BaseX JAR archive
5. Choose "org.basex.BaseXClient" as "Main class"
6. Set interpreter arguments for your server, port, username and password, e.g. -Uadmin -Padmin -nlocalhost -p1984.

Usage

The query execution works as follows:

1. Create a new XQuery Project with *File* → *New* → *XQuery Project*.
2. Add a new XQuery Module with *File* → *New* → *XQuery Module*.
3. Edit your XQuery Module and execute it with *Run*.
4. The results are displayed in the Console window of Eclipse.

References

- [1] <http://www.eclipse.org>

Query Features

XQuery

Welcome to the Query Portal, which is one of the Main Sections of this documentation. BaseX provides an implementation of the W3 XPath ^[1] and XQuery ^[2] languages, which are tightly coupled with the underlying database store. However, the processor is also a flexible general purpose processor, which can access local and remote sources. High conformance with the official specifications is one of our main objectives, as the results of the XQuery Test Suite ^[3] demonstrate. This section contains information on the query processor and its extensions:

XQuery 3.0

Features of the upcoming XQuery 3.0 ^[4] Recommendation.

Module Library

Additional functions included in the internal modules.

Repository

Install and manage XQuery and Java modules.

Java Bindings

Accessing and calling Java code from XQuery.

Full-Text

How to use BaseX as a full-fledged full-text processor.

Updates

Updating databases and local resources via XQuery Update.

Serialization

Serialization parameters supported by BaseX.

Errors

Errors raised by XQuery expressions.

References

[1] <http://www.w3.org/TR/xpath-30/>

[2] <http://www.w3.org/TR/xquery-30>

[3] http://dev.w3.org/2006/xquery-test-suite/PublicPagesStagingArea/XQTSReportSimple_XQTS_1_0_2.html

[4] <http://www.w3.org/TR/xquery-30/>

XQuery 3.0

This article is part of the XQuery Portal. It summarizes features of the upcoming XQuery 3.0 ^[4] and XPath 3.0 ^[1] Recommendations that have already been implemented in BaseX.

Group By

FLWOR expressions have been extended to include the `group by` ^[1] clause, which is well-established among relational database systems. `group by` can be used to apply value-based partitioning to query results:

Example:

```
for $ppl in doc('xmark')//people/person
let $ic := $ppl/profile/@income
let $income := if($ic < 30000) then
    "challenge"
    else if($ic >= 30000 and $ic < 100000) then
    "standard"
    else if($ic >= 100000) then
    "preferred"
    else
    "na"
group by $income
order by $income
return element { $income } { count($ppl) }
```

This query is a rewrite of Query #20 ^[2] contained in the XMark Benchmark Suite ^[3] to use `group by`. The query partitions the customers based on their income.

Result:

```
<challenge>4731</challenge>
<na>12677</na>
<prefered>314</prefered>
<standard>7778</standard>
```

In contrast to the relational GROUP BY statement, the XQuery counterpart concatenates the values of all non-grouping variables that belong to a specific group. In the context of our example, all nodes in `//people/person` that belong to the "preferred" partition are concatenated in `$ppl` after grouping has finished. You can see this effect by changing the return statement to:

```
...
return element { $income } { $ppl }
```

Result:

```
<challenge>
  <person id="person0">
    <name>Kasidit Treweek</name>
    ...
  <person id="personX">
    ...
```

```
</challenge>
```

Try/Catch

The try/catch ^[4] construct can be used to handle errors at runtime:

Example:

```
try {
  1 + '2'
} catch XPTY0004 {
  'Typing error: ' || $err:description
} catch * {
  'Error [' || $err:code || ']: ' || $err:description
}
```

Result: Typing error: '+' operator: number expected, xs:string found.

Switch

The switch ^[5] statement is available in many other programming languages. It chooses one of several expressions to evaluate based on its input value.

Example:

```
for $fruit in ("Apple", "Pear", "Peach")
return switch ($fruit)
  case "Apple" return "red"
  case "Pear"   return "green"
  case "Peach" return "pink"
  default      return "unknown"
```

Result: red green pink

Function Items

One of the most distinguishing features added in *XQuery 3.0* are *function items*, also known as *lambdas* or *lambda functions*. They make it possible to abstract over functions and thus writing more modular code.

Examples:

Function items can be obtained in three different ways:

- Declaring a new *inline function*:

```
let $f := function($x, $y) { $x + $y }
return $f(17, 25)
```

Result: 42

- Getting the function item of an existing (built-in oder user-defined) XQuery function. The arity (number of arguments) has to be specified as there can be more than one function with the same name:

```
let $f := math:pow#2
return $f(5, 2)
```

Result: 25

- *Partially applying* another function or function item. This is done by supplying only some of the required arguments, writing the placeholder `?` in the positions of the arguments left out. The produced function item has one argument for every placeholder.

```
let $f := fn:substring(?, 1, 3)
return (
  $f('foo123'),
  $f('bar456')
)
```

Result: foo bar

Function items can also be passed as arguments to and returned as results from functions. These so-called Higher-Order Functions like `fn:map` and `fn:fold-left` are discussed in more depth on their own Wiki page.

Expanded QName

Updated with Version 7.2:

A *QName* can now be directly prefixed with the letter "Q" and a namespace URI in the Clark Notation ^[1].

Examples:

- `Q{http://www.w3.org/2005/xpath-functions/math}pi()` returns the number π
- `Q{java:java.io.FileOutputStream}new("output.txt")` creates a new Java file output stream

Before [Version 7.2](#), and in older versions of the XQuery 3.0 specification, the prefixed namespace URI was quoted:

- `"http://www.w3.org/2005/xpath-functions/math":pi()`
- `"java:java.io.FileOutputStream":new("output")`

Namespace Constructors

New namespaces can now be created via so-called 'Computed Namespace Constructors'.

```
element node { namespace pref { 'http://url.org/' } }
```

String Concatenations

Two vertical bars `||` (also names *pipe characters*) can be used to concatenate strings. This operator is a shortcut for the `fn:contains()` function.

```
'Hello' || ' ' || 'Universe'
```

External Variables

Default values can now be attached to external variable declarations. This way, an expression can also be evaluated if its external variables have not been bound to a new value.

```
declare variable $user external := "admin";
>User:", $user
```

Serialization

Serialization parameters can now be defined within XQuery expressions. Parameters are placed in the query prolog and need to be specified as option declarations, using the `output` prefix.

Example:

```
declare option output:omit-xml-declaration "no";
declare option output:method "xhtml";
<html/>
```

Result: <?xml version="1.0" encoding="UTF-8"?><html></html>

Context Item

The context item can now be specified in the prolog of an XQuery expressions:

Example:

```
declare context item := document {
  <xml>
    <text>Hello</text>
    <text>World</text>
  </xml>
};

for $t in .//text()
return string-length($t)
```

Result: 5 5

Annotations

Introduced with Version 7.2:

XQuery 3.0 introduces annotations to declare properties associated with functions and variables. For instance, a function may be declared `%public`, `%private`, or `%updating`.

Example:

```
declare %private function local:max($x1, $x2) {
  if($x1 > $x2) then $x1 else $x2
};

local:max(2, 3)
```

Functions

BaseX supports all functions that have been added in Version 3.0 of the XQuery Functions and Operators ^[6] Working Draft. The new functions are listed below:

- `math:pi()`, `math:sin()`, and many others (see Math Module)
- `fn:analyze-string()`
- `fn:available-environment-variables()`
- `fn:element-with-id()`
- `fn:environment-variable()`
- `fn:filter()`
- `fn:fold-left()`
- `fn:fold-right()`
- `fn:format-date()`
- `fn:format-dateTime()`

- `fn:format-integer()`
- `fn:format-number()`
- `fn:format-time()`
- `fn:function-arity()`
- `fn:function-lookup()`
- `fn:function-name()`
- `fn:generate-id()`
- `fn:has-children()`
- `fn:head()`
- `fn:innermost()`
- `fn:map()`
- `fn:map-pairs()`
- `fn:outermost()`
- `fn:parse-xml()`
- `fn:path()`
- `fn:serialize()`
- `fn:tail()`
- `fn:unparsed-text()`
- `fn:unparsed-text-available()`
- `fn:unparsed-text-lines()`
- `fn:uri-collection()`

New signatures have been added for the following functions:

- `fn:document-uri()` with 0 arguments
- `fn:string-join()` with 1 argument
- `fn:node-name()` with 0 arguments
- `fn:round()` with 2 arguments
- `fn:data()` with 0 arguments

Changelog

Version 7.2

- Added: Annotations
- Updated: EQName syntax

Version 7.1

- Added: Expanded QNames, Computed Namespace Constructor

Version 7.0

- String Concatenator

References

- [1] <http://www.w3.org/TR/xquery-30/#id-group-by>
- [2] <http://www.ins.cwi.nl/projects/xmark/Assets/xmlquery.txt>
- [3] <http://www.ins.cwi.nl/projects/xmark>
- [4] <http://www.w3.org/TR/xquery-30/#id-try-catch>
- [5] <http://www.w3.org/TR/xquery-30/#id-switch>

[6] <http://www.w3.org/TR/xpath-functions-30/>

Higher-Order Functions

This page talks about *higher-order functions* introduced with XQuery 3.0. The BaseX-specific `hof` module containing some more very useful functions can be found at [Higher-Order Functions Module](#).

Function Items

Probably the most important new feature in XQuery 3.0 are *function items*, i. e. items that act as functions, but can also be passed to and from other functions and expressions, making functions *first-class citizens* of the language.

The XQuery 3.0 page goes into details on how function items can be obtained.

Function Types

Like every XQuery item, function items have a *sequence type*. It can be used to specify the *arity* (number of arguments the function takes) and the argument and result types.

The most general function type is `function(*)`. It's the type of all function items. The following query for example goes through a list of XQuery items and, if it is a function item, prints its arity:

```
for $item in (1, 'foo', fn:concat#3, function($a) { 42 * $a })
where $item instance of function(*)
return fn:function-arity($item)
```

Result: 3 1

The notation for specifying argument and return types is quite intuitive, as it closely resembles the function declaration. The XQuery function

```
declare function local:char-at(
  $str as xs:string,
  $pos as xs:integer
) as xs:string {
  fn:substring($str, $pos, 1)
};
```

for example has the type `function(xs:string, xs:integer) as xs:string`. It isn't possible to specify only the argument and not the result type or the other way round. A good place-holder to use when no restriction is wanted is `item()*`, as it matches any XQuery value.

Function types can also be nested. As an example we take `local:on-sequences`, which takes a function defined on single items and makes it work on sequences as well:

```
declare function local:on-sequences(
  $f as function(item()) as item()*
) as function(item()* as item()* {
  fn:map($f, ?)
};
```

We'll see later how `fn:map(...)` works. The type of `local:on-sequences(...)` on the other hand is easily constructed, if a bit long:

```
function(function(item()) as item(*) as function(item(*) as item(*)
```

Higher-Order Functions

A *higher-order function* is a function that takes other functions as arguments and/or returns them as results. `fn:map` and `local:on-sequences` from the last chapter are nice examples.

With the help of higher-order functions, one can extract common patterns of *behaviour* and abstract them into a library function.

Higher-Order Functions on Sequences

Some usage patterns on sequences are so common that the higher-order functions describing them are in the XQuery standard libraries. They are listed here, together with their possible XQuery implementation and some motivating examples.

fn:map(\$f, \$seq)

Signatures `fn:map($f as function(item()) as item(*), $seq as item(*) as item(*)`

Summary Applies the function `item $f` to every element of the sequence `$seq` and returns all of the results as a sequence.

Examples

- Squaring all numbers from 1 to 10:

```
fn:map(math:pow(?, 2), 1 to 10)
```

Result: 1 4 9 16 25 36 49 64 81 100

- Applying a list of functions to a string:

```
let $fs := (
  fn:upper-case#1,
  fn:substring(?, 4),
  fn:string-length#1
)
return fn:map(function($f) { $f('foobar') }, $fs)
```

Result: FOOBAR bar 6

XQuery 1.0

```
declare function local:map(
  $f as function(item()) as item(*),
  $seq as item(*)
) as item(*) {
  for $x in $seq
  return $f($seq)
};
```

fn:filter(\$pred, \$seq)

Signatures `fn:filter`(\$pred as function(item()) as xs:boolean, \$seq as item()*) as item()*

Summary Applies the boolean predicate \$pred to all elements of the sequence \$seq, returning those for which it returns true().

Examples

- All even integers until 10:

```
fn:filter(function($x) { $x mod 2 eq 0 }, 1 to 10)
```

Result: 2 4 6 8 10

- Strings that start with an upper-case letter:

```
let $first-upper := function($str) {
  let $first := fn:substring($str, 1, 1)
  return $first eq fn:upper-case($first)
}
return fn:filter($first-upper, ('FooBar', 'foo', 'BAR'))
```

Result: FooBar BAR

- Inefficient prime number generator:

```
let $is-prime := function($x) {
  $x gt 1 and (every $y in 2 to ($x - 1) satisfies $x mod $y ne 0)
}
return filter($is-prime, 1 to 20)
```

Result: 2 3 5 7 11 13 17 19

Note `fn:filter` can be easily implemented with `fn:map`:

```
declare function local:filter($pred, $seq) {
  map(
    function($x) {
      if($pred($x)) then $x else ()
    },
    $seq
  )
};
```

XQuery 1.0

```
declare function local:filter(
  $pred as function(item()) as xs:boolean,
  $seq as item()*
) as item()* {
  $seq[$pred(.)]
};
```

fn:map-pairs(\$f, \$seq1, \$seq2)

Signatures `fn:map-pairs`(\$f as function(item(), item()) as item()*, \$seq1 as item()*, \$seq2 as item()*) as item()*

Summary *zips* the elements from the two sequences \$seq1 and \$seq2 together with the function \$f. It stops after the shorter sequence ends.

Examples • Adding one to the numbers at odd positions:

```
fn:map-pairs(
  function($a, $b) { $a + $b },
  fn:map(function($x) { $x mod 2 }, 1 to 10),
  (1, 1, 1, 1, 1)
)
```

Result: 2 1 2 1 2

• Line numbering:

```
let $number-lines := function($str) {
  fn:string-join(
    fn:map-pairs(
      concat(?, ': ', ?),
      1 to 1000,
      tokenize($str, '\r?\n|\r')
    ),
    '&#xa;'
  )
}
return $number-lines(
  'hello world,
  how are you?'
)
```

Result:

```
1: hello world,
2: how are you?
```

• Checking if a sequence is sorted:

```
let $is-sorted := function($seq) {
  every $b in
    fn:map-pairs(
      function($a, $b) { $a le $b },
      $seq,
      fn:tail($seq)
    )
  satisfies $b
}
return (
  $is-sorted(1 to 10),
  $is-sorted((1, 2, 42, 4, 5))
)
```

Result: true false**XQuery**
1.0

```
declare function local:map-pairs(
  $f as function(item(), item()) as item()*,
  $seq1 as item()*,
  $seq2 as item()*
) as item()* {
  for $pos in 1 to min(length($seq1), length($seq2))
  return $f($seq1[$pos], $seq2[$pos])
};
```

Folds

A *fold*, also called *reduce* or *accumulate* in other languages, is a very basic higher-order function on sequences. It starts from a seed value and incrementally builds up a result, consuming one element from the sequence at a time and combining it with the aggregate with a user-defined function.

Folds are one solution to the problem of not having *state* in functional programs. Solving a problem in *imperative* programming languages often means repeatedly updating the value of variables, which isn't allowed in functional languages.

Calculating the *product* of a sequence of integers for example is easy in Java:

```
public int product(int[] seq) {
    int result = 1;
    for(int i : seq) {
        result = result * i;
    }
    return result;
}
```

Nice and efficient implementations using folds will be given below.

The *linear* folds on sequences come in two flavours. They differ in the direction in which they traverse the sequence:

fn:fold-left(\$f, \$seed, \$seq)

Signatures `fn:fold-left` (\$f as function(item()*), item()) as item()*, \$seed as item()*, \$seq as item()*) as item()*

Summary The *left fold* traverses the sequence from the left. The query `fn:fold-left($f, 0, 1 to 5)` for example would be evaluated as:

```
$f($f($f($f($f(0, 1), 2), 3), 4), 5)
```

Examples • Product of a sequence of integers:

```
let $product := fn:fold-left(
  function($result, $i) { $result * $i },
  1,
  ?
)
return $product(1 to 5)
```

Result: 120

• Illustrating the evaluation order:

```
fn:fold-left(
  concat('$f(', ?, ', ', '?', ')'),
  '$seed',
  1 to 5
)
```

Result: \$f(\$f(\$f(\$f(\$f(\$seed, 1), 2), 3), 4), 5)

• Building a decimal number from digits:

```
let $from-digits := fold-left(
  function($n, $d) { 10 * $n + $d },
  0,
  ?
)
return (
  $from-digits(1 to 5),
  $from-digits((4, 2))
)
```

Result: 12345 42

XQuery As folds are more general than *FLWOR* expressions, the implementation isn't as concise as the former ones:

1.0

```
declare function local:fold-left(
  $f as function(item()*, item()) as item()*,
  $seed as item()*,
  $seq as item()*
) as item()* {
  if(empty($seq) then $seed
  else local:fold-left(
    $f,
    $f($seed, fn:head($seq)),
    fn:tail($seq)
  )
};
```

fn:fold-right(\$f, \$seed, \$seq)

Signatures `fn:fold-right`(\$f as function(item(), item()*) as item()*, \$seed as item()*, \$seq as item()*) as item()*

Summary The *right fold* `fn:fold-right`(\$f, \$seed, \$seq) traverses the from the right. The query `fn:fold-right`(\$f, 0, 1 to 5) for example would be evaluated as:

```
$f(1, $f(2, $f(3, $f(4, $f(5, 0))))
```

Examples

- Product of a sequence of integers:

```
let $product := fn:fold-right(
  function($i, $result) { $result * $i },
  1,
  ?
)
return $product(1 to 5)
```

Result: 120

- Illustrating the evaluation order:

```
fn:fold-right(
  concat('$f(', '?', ', ', '?', ')'),
  '$seed',
  1 to 5
)
```

Result: \$f(1, \$f(2, \$f(3, \$f(4, \$f(5, \$seed))))

- Reversing a sequence of items:

```
let $reverse := fn:fold-right(
  function($item, $rev) {
    $rev, $item
  },
  (),
  ?
)
return $reverse(1 to 10)
```

Result: 10 9 8 7 6 5 4 3 2 1

XQuery
1.0

```
declare function local:fold-right(
  $f as function(item(), item()*) as item()*,
  $seed as item()*,
  $seq as item()*
) as item()* {
  if(empty($seq)) then $seed
  else $f(
    fn:head($seq),
    local:fold-right($f, $seed, tail($seq))
  )
};
```

Note that the order of the arguments of `$f` are inverted compared to that in `fn:fold-left`(...).

Module Library

This article is part of the XQuery Portal.

Beside the standard XQuery Functions ^[6], BaseX offers additional function modules, which are listed in the following table. All modules are statically bound, which means that they need not (but may) be explicitly declared in the query prolog.

Module	Description	Prefix	Namespace URI
Cryptography	Cryptographic functions, based on the EXPath Cryptographic ^[1] module.	crypto	http://expath.org/ns/crypto
Database	Functions for accessing and updating databases.	db	http://basex.org/modules/db
File	File handling, based on the latest draft of the EXPath File ^[2] module.	file	http://expath.org/ns/file
Full-Text	Functions for performing full-text operations.	ft	http://basex.org/modules/ft
Higher-Order	Additional higher-order functions that are not in the standard libraries.	hof	http://basex.org/modules/hof
HTTP	Sending HTTP requests, based on the EXPath HTTP ^[3] module.	http	http://expath.org/ns/http-client
Index	Functions for requesting details on database indexes. Version 7.1	index	http://basex.org/modules/index
JSON	Parsing and serializing JSON documents ^[4] .	json	http://basex.org/modules/json
Map	Functions for handling maps (key/value pairs).	map	http://www.w3.org/2005/xpath-functions/map
Math	Mathematical operations, extending the W3C Working Draft ^[6] .	math	http://www.w3.org/2005/xpath-functions/math
Repository	Installing, deleting and listing packages. Version 7.1	repo	http://basex.org/modules/repo
SQL	JDBC bridge to access relational databases.	sql	http://basex.org/modules/sql
Utility	Utility functions, used for data conversions, profiling and dynamic evaluation.	util	http://basex.org/modules/util
XSLT	Stylesheet transformations, based on Java's and Saxon's XSLT processor.	xslt	http://basex.org/modules/xslt
ZIP	ZIP functionality, based on the EXPath ZIP ^[5] module.	zip	http://expath.org/ns/zip

References

- [1] <http://expath.org/spec/crypto>
- [2] <http://expath.org/spec/file>
- [3] <http://expath.org/spec/http-client>
- [4] <http://www.json.org>
- [5] <http://expath.org/spec/zip>

Repository

This article is part of the XQuery Portal. It describes how external XQuery modules and Java code can be installed in the XQuery module repository, and how new packages are built and deployed.

Motivation

One of the reasons why languages such as Java or Perl have been so successful is the vast amount of libraries that are available to developers. XQuery is a Turing complete language, but it just provides around 100 pre-defined functions, which cannot meet all requirements. This is why additional libraries arise – such as FunctX ^[1] – that extend the language with new features.

BaseX offers two mechanisms to make new packages accessible to the XQuery processor:

1. With [Version 7.2.1](#), we offer a simple packaging mechanism to directly install single XQuery and Java modules to the repository.
2. The EXPath Packaging system provides a generic mechanism for adding XQuery modules to query processors. A package is defined as a `.xar` archive, which encapsulates one or more extension libraries.

Usage

All packages are stored in the *repository*. The repository is a directory named `BaseXRepo` or `repo`, which resides in your home directory. BaseX provides three commands for interaction with the package repository: `REPO INSTALL`, `REPO DELETE`, and `REPO LIST`. Packages can also be managed from within XQuery, using the Repository Module.

Installation

A module or package can be installed with the `REPO INSTALL` command. The path to the file has to be given as a parameter, as the following two examples demonstrate:

```
REPO INSTALL http://files.basex.org/modules/functx-1.0.xar
REPO INSTALL hello-world.xqm
```

The installation will only succeed if the specified file conforms to the constraints described below. If you know that your input is valid, you may as well copy the files directly to the repository directory, or edit its contents in the repository without deleting and reinstalling them.

Since [Version 7.2.1](#), existing packages are simply replaced (before, an error was raised).

Querying

Installed packages can be addressed by importing them as *modules*. Since we have the package repository in which all packages are located, it is sufficient to just specify the namespace URI of a module:

```
import module namespace functx = "http://www.functx.com";
```

When this statement is parsed, the query processor will check if the namespace "http://www.functx.com" is used in any of the installed packages and, if yes, will load and parse the modules. In the remaining query, you can call the parsed module functions in the standard way, e.g.:

```
functx:capitalize-first("test")
```

Package encapsulating Java archives can be imported in the same way as pure XQuery modules (see below).

Listing

All currently installed packages can be listed with the `REPO LIST` command. It will return the names of all packages, their version, and the directory in which they are installed:

```
URI                Version  Directory
-----
http://www.functx.com  1.0      http-www.functx.com-1.0

1 package(s).
```

Removal

A package can be deleted with the command `REPO DELETE` and by specifying its name or (since [Version 7.2.1](#)) the name, suffixed with a hyphen and the package version:

```
REPO DELETE http://www.functx.com ...or...
REPO DELETE http://www.functx.com-1.0
```

Packaging

With [Version 7.2.1](#), XQuery modules and JAR archives can directly be added to the repository without further packaging efforts:

XQuery

If an XQuery file is specified as input for the `install` command, it will be parsed as XQuery module. If parsing was successful, the module URI will be rewritten to a file path and attached with the `.xqm` file suffix, and the original file will be renamed and copied to that path into the repository.

Example:

Contents of the file `HelloWorld.xqm` ^[2] (comments removed):

```
module namespace m = 'http://basex.org/modules/Hello';
declare function m:hello($world) {
  'Hello ' || $world
};
```

Installation (the original file will be copied to `org/basex/modules/Hello.xqm`):

```
REPO INSTALL HelloWorld.xqm
```

XQuery file `HelloUniverse.xq` ^[2] (comments removed):

```
import module namespace m = 'http://basex.org/modules/Hello';
m:hello("Universe")
```

Java

Suitable JAR archives may contain one or more class files. One of them will be chosen as main class, which must be specified in a `Main-Class` entry in the manifest file (`META-INF/MANIFEST.MF`). This fully qualified Java class name will be rewritten to a file path by replacing the dots with slashes and attaching with the `.jar` file suffix, and the original file will be renamed and copied to that path into the repository.

The public functions of this class can then be addressed from XQuery, using the class or file path as namespace URI, or an alternative writing that can be rewritten to the module file path. Moreover, a class may extend the `QueryModule` class to get access to the current query context and to be enriched by some helpful annotations (please consult Context Awareness of Java Bindings for more information).

Example:

Structure of the `HelloWorld.jar`^[3] archive:

```
META-INF/
  MANIFEST.MF
org/basex/modules/
  Hello.class
```

Contents of the file `MANIFEST.mf` (the whitespaces are obligatory):

```
Manifest-Version: 1.0
Main-Class: org.basex.modules.Hello
```

Contents of the file (comments removed):

```
package org.basex.modules;
public class Hello {
    public String hello(final String world) {
        return "Hello " + world;
    }
}
```

Installation (the file will be copied to `org/basex/modules/Hello.jar`):

```
REPO INSTALL HelloWorld.jar
```

XQuery file `HelloUniverse.xq`^[2] (same as above):

```
import module namespace m = 'http://basex.org/modules/Hello';
m:hello("Universe")
```

After installing the module, all of the following URIs can be used in XQuery to import this module or call its functions:

```
http://basex.org/modules/hello/World
org/basex/modules/hello/World
org.basex.modules.hello.World
```

Please be aware that the execution of Java code can cause side effects that conflict with the functional nature of XQuery, or may introduce new security risks. The article on Java Bindings gives more insight on how Java code is handled from the XQuery processor.

EXPath Packaging

The EXPath specification ^[4] defines how the structure of a `.xar` archive shall look like. The package contains at its root a package descriptor named `expath-pkg.xml`. This descriptor presents some meta data about the package as well as the libraries which it contains and their dependencies on other libraries or processors.

XQuery

Apart from the package descriptor, a `.xar` archive contains a directory which includes the actual XQuery modules. For example, the FunctX XQuery Library ^[5] is packaged as follows:

```
expath-pkg.xml
functx/
  functx.xql
  functx.xsl
```

Java

In case you want to extend BaseX with a Java archive, some additional requirements have to be fulfilled:

- Apart from the package descriptor `expath-pkg.xml`, the package has to contain a descriptor file at its root, defining the included jars and the binary names of their public classes. It must be named `basex.xml` and must conform to the following structure:

```
<package xmlns="http://expath.org/ns/pkg">
  <jar>...</jar>
  ....
  <class>...</class>
  <class>...</class>
  ....
</package>
```

- The jar file itself along with an XQuery file defining wrapper functions around the java methods has to reside in the module directory. The following example illustrates how java methods are wrapped with XQuery functions:

Example:

Suppose we have a simple class `Printer` having just one public method `print()`:

```
package test;

public final class Printer {
  public String print(final String s) {
    return new Writer(s).write();
  }
}
```

We want to extend BaseX with this class and use its method. In order to make this possible we have to define an XQuery function which wraps the `print` method of our class. This can be done in the following way:

```
import module namespace j="http://basex.org/lib/testJar";

declare namespace p="java:test.Printer";

declare function j:print($str as xs:string) as xs:string {
```

```
let $printer := p:new()
return p:print($printer, $str)
};
```

As it can be seen, the class `Printer` is declared with its binary name as a namespace prefixed with "java" and the XQuery function is implemented using the Java Bindings^[6] offered by BaseX.

On our file server^[7], you can find some example libraries packaged as XML archives (xar files). You can use them to try our packaging API or just as a reference for creating your own packages.

URI Rewriting

If modules are looked up in the repository, their URIs are rewritten to a local file path. The URI transformation has been inspired by Zorba^[8]:

1. If a URI authority exists, it is reversed, and its dots are replaced by slashes.
2. The URI path is appended. If no path exists, a single slash is appended instead.
3. If the resulting string ends with a slash, the `index` string is appended.

If the resulting path has no file suffix, it may point to either an XQuery module or a Java archive. The following examples show some rewritings:

- `http://basex.org/modules/hello/World` → `org/basex/modules/hello/World`
- `http://www.example.com` → `com/example/www/index`
- `a/little/example.xq` → `a/little/example.xq`

Changelog

Version 7.2.1

- Updated: Installation: existing packages will be replaced
- Updated: Removal: remove specific version of a package
- Added: Packaging, URI Rewriting

Version 7.1

- Added: Repository Module

Version 7.0

- Added: EXPath Packaging

References

- [1] <http://www.xqueryfunctions.com/>
- [2] <http://files.basex.org/modules/hello/HelloWorld.xqm>
- [3] <http://files.basex.org/modules/hello/HelloWorld.jar>
- [4] <http://expath.org/spec/pkg>
- [5] <http://www.functx.com/>
- [6] http://docs.basex.org/wiki/Java_Bindings
- [7] <http://files.basex.org/modules/>
- [8] <http://www.zorba-xquery.com/html/documentation/latest/zorba/uriresolvers>

Java Bindings

This article is part of the XQuery Portal. It demonstrates two ways to invoke Java code from XQuery, and (since [Version 7.2.1](#)) an extension to make Java code aware of the current context.

The Java Binding feature is an extensibility mechanism which enables developers to directly access Java variables and execute code from XQuery. Java classes are identified by namespaces. The namespace URI must simply contain the fully qualified class name. The URI can optionally be prefixed with the string `java:` to enforce that the addressed code is written in Java.

If the addressed Java code is not found in the classpath, it first needs to be installed in the Repository.

Namespace Declarations

Java classes can be declared via namespaces. The namespace can then be used to call static functions contained in that class.

The following example uses Java's `Math` class to return the cosine of an angle by calling the static method `cos()`:

```
declare namespace math = "java:java.lang.Math";
math:cos(xs:double(0))
```

The new Expanded QName notation of XQuery 3.0 can be applied as well to directly specify a namespace URI instead of the prefix:

```
Q{java.lang.Math}cos(xs:double(0))
```

The constructor of a class can be invoked by calling the virtual function `new()`. Instance methods can then be called by passing on the resulting Java object as first argument.

The following example writes 256 bytes to the file `output.txt`. First, a new `FileWriter` instance is created, and its `write()` function is called in the next step:

```
declare namespace fw = "java.io.FileWriter";
let $file := fw:new('output.txt')
return (
  for $i in 0 to 255
  return fw:write($file, xs:int($i)),
  fw:close($file)
)
```

In general, it is recommended to use XQuery expressions and functions whenever possible, as Java code cannot be pre-compiled, and will often be evaluated slower than optimized XQuery code. Next, Java code can only be executed with admin permissions.

Module Imports

Introduced with Version 7.2.1:

Java code can also be integrated by *importing* classes as modules. A new instance of the addressed class is created, which can then be accessed in the query body.

An example (the boolean values returned by `set:add()` are ignored):

```
import module namespace set = "java.util.HashSet";
let $loop :=
  for $i in 1 to 10000
  return set:add($i)
return set:size()
```

Advantages of this approach are:

- imported code can be executed faster than instances created at runtime via `new()`.
- the work on class instances ensures that queries run in parallel will not cause any concurrency issues (provided that the class contains no static variables or functions).

A drawback is that no arguments can be passed on to the class constructor. This is also why the class must provide a constructor without no arguments.

Context-Awareness

Introduced with Version 7.2.1:

Context-aware classes represent a powerful interface for writing Java modules that are more closely coupled with the BaseX core library. If an instantiated class inherits the abstract `QueryModule`^[1] class of BaseX, it will get access to the `context` variable, which is an instance of the `QueryContext`^[2] class. It provides access to all static and dynamic properties of the current query. Additionally, the default properties of functions can be changed via annotations:

- Java functions can only be executed by users with Admin permissions. You may annotate a function with `@Requires(<Permission>)` to also make it accessible to users with less privileges.
- Java code is treated as *non-deterministic*, as its behavior cannot be predicted by the XQuery processor. You may annotate a function as `@Deterministic` if you know that it will have no side-effects and will always yield the same result.
- Java code is treated as *context-independent*. If a function accesses the query context, it should be annotated as `@ContextDependent`
- Java code is treated as *focus-independent*. If a function accesses the current context item, position or size, it should be annotated as `@FocusDependent`

The following XQuery code invokes two Java methods. The first Java function retrieves information from the static query context, and the second one throws a query exception:

```
import module namespace context = 'org.basex.examples.query.ContextModule';

<context>{
  context:function-namespace()
}</context>,
<to-int>{
  try { context:to-int('abc') }
  catch * { 'Error in line', $err:line-number }
```

```
}</to-int>
```

The imported Java class is shown below:

```
package org.basex.examples.query;

import org.basex.query.*;
import org.basex.query.item.*;
import org.basex.util.*;

/**
 * This example is inherited from the {@link QueryModule} class.
 */
public class ContextModule extends QueryModule {
    /**
     * Returns the default function namespace.
     * @return default function namespace
     */
    @Requires(Permissions.NONE)
    @Deterministic
    @ContextDependent
    public Str functionNS() {
        return Str.get(context.sc.nsFunc);
    }

    /**
     * Converts the specified string to an integer.
     * @param value string representation
     * @return integer
     * @throws QueryException query exception
     */
    @Requires(Permissions.NONE)
    @Deterministic
    public int toInt(final String value) throws QueryException {
        try {
            return Integer.parseInt(value);
        } catch (NumberFormatException ex) {
            throw new QueryException(ex.getMessage());
        }
    }
}
```

The result will look as follows:

```
<context>http://www.w3.org/2005/xpath-functions</context>
<to-int>Error in line 6</to-int>
```

Please visit the XQuery 3.0 specification if you want to get more insight into function properties ^[3].

Changelog

Version 7.2.1

- Added: import of Java modules, context awareness

References

- [1] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/query/QueryModule.java>
- [2] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/query/QueryContext.java>
- [3] <http://www.w3.org/TR/xpath-functions-30/#properties-of-functions>

Full-Text

This article is part of the XQuery Portal. It summarizes the full-text features of BaseX.

Full-text retrieval is an essential query feature for working with XML documents, and BaseX was the first query processor that fully supported the W3C XQuery Full Text 1.0 ^[1] Recommendation. This page lists some singularities and extensions of the BaseX implementation.

Query Evaluation

BaseX offers different evaluation strategies for XQFT queries, the choice of which depends on the input data and the existence of a full text index. The query compiler tries to optimize and speed up queries by applying a full text index structure whenever possible and useful. Three evaluation strategies are available: the standard sequential database scan, a full-text index based evaluation and a hybrid one, combining both strategies (see "XQuery Full Text implementation in BaseX" ^[2]). Query optimization and selection of the most efficient evaluation strategy is done in a full-fledged automatic manner. The output of the query optimizer indicates which evaluation plan is chosen for a specific query. It can be inspected by activating verbose querying (Command: `SET VERBOSE ON`) or opening the Query Info in the GUI. The message

```
Applying full-text index
```

suggests that the full-text index is applied to speed up query evaluation. A second message

```
Removing path with no index results
```

indicates that the index does not yield any results for the specified term and is thus skipped. If index optimizations are missing, it sometimes helps to give the compiler a second chance and try different rewritings of the same query.

Options

The available full-text index can handle various combinations of the match options defined in the XQuery Full Text Recommendation. By default, most options are disabled. The GUI dialogs for creating new databases or displaying the database properties contain a tab for choosing between all available options. On the command-line, the `SET` command can be used to activate full-text indexing or creating a full-text index for existing databases:

- `SET FTINDEX true; CREATE DB input.xml`
- `CREATE INDEX fulltext`

The following indexing options are available:

- **Language:** see below for more details (`SET LANGUAGE EN`)
 - **Support Wildcards:** a trie-based index can be applied to support wildcard searches (`SET WILDCARDS true`)
 - **Stemming:** tokens are stemmed with the Porter Stemmer before being indexed (`SET STEMMING true`)
-

- **Case Sensitive:** tokens are indexed in case-sensitive mode (`SET CASESENS true`)
- **Diacritics:** diacritics are indexed as well (`SET DIACRITICS true`)
- **TF/IDF Scoring:** TF/IDF-based scoring values are calculated and stored in the index (`SET SCORING 0/1/2`; details see below)
- **Stopword List:** a stop word list can be defined to reduce the number of indexed tokens (`SET STOPWORDS [filename]`)

Languages

The chosen language determines how the input text will be tokenized and stemmed. The basic code base and jar file of BaseX comes with built-in support for English and German. More languages are supported if the following libraries are found in the classpath:

- `lucene-stemmers-3.4.0.jar` ^[3]: includes Snowball and Lucene stemmers and extends language support to the following languages: Arabic, Bulgarian, Catalan, Czech, Danish, Dutch, Finnish, French, Hindi, Hungarian, Italian, Latvian, Lithuanian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish.
- `igo-0.4.3.jar` ^[4]: An additional article explains how Igo can be integrated, and how Japanese texts are tokenized and stemmed.

The JAR files can also be found in the `zip` and `exe` distribution files of BaseX.

The following two queries, which both return `true`, demonstrate that stemming depends on the selected language:

```
"Indexing" contains text "index" using stemming,
"häuser" contains text "haus" using stemming using language "de"
```

Scoring

The XQuery Full Text Recommendation allows for the usage of scoring models and values within queries, with scoring being completely implementation defined. BaseX offers an efficient internal scoring model which can be easily extended to different application scenarios. Additionally, BaseX allows to store scoring values within the full-text index structure (demanding additional time and memory). Three scoring types are currently available, which can be adjusted with the `SCORING` property (Default: `SET SCORING 0`):

- `0`: This algorithm yields the best results for general-purpose use cases. It calculates the scoring value out of the length of a term and its frequency in a single text node. This algorithm is also applied if no index exists, or if the index cannot be applied in a query.
- `1`: Standard TF/IDF algorithm, which treats *document nodes* as document units.
- `2`: Each *text node* is treated as a document unit in the TF/IDF algorithm. This variant is an alternative for type 1, if the database contains large, few XML files.

Querying Using Thesaurus

BaseX supports full-text queries using thesauri, but it does not provide a default thesaurus. This is why query such as

```
'computers' contains text 'hardware'
using thesaurus default
```

will return `false`. However, if the thesaurus is specified, then the result will be `true`

```
'computers' contains text 'hardware'
using thesaurus at 'XQFTTS_1_0_4/TestSources/usability2.xml'
```

The format of the thesaurus files must be the same as the format of the thesauri provided by the XQuery and XPath Full Text 1.0 Test Suite ^[5]. It is an XML with structure defined by an XSD Schema ^[6].

Fuzzy Querying

In addition to the official recommendation, BaseX supports fuzzy querying. The XQFT grammar was enhanced by the `FTMatchOption` using `fuzzy` to allow for approximate searches in full texts. By default, the standard full-text index already supports the efficient execution of fuzzy searches.

Document 'doc.xml':

```
<doc>
  <a>house</a>
  <a>hous</a>
  <a>haus</a>
</doc>
```

Command: `CREATE DB doc.xml; CREATE INDEX fullext`

Query:

```
//a[text() contains text 'house' using fuzzy]
```

Result:

```
<a>house</a>
<a>hous</a>
```

Fuzzy search is based on the Levenshtein distance. The maximum number of allowed errors is calculated by dividing the token length of a specified query term by 4, preserving a minimum of 1 errors. A static error distance can be set by adjusting the `LSERROR` property (default: `SET LSERROR 0`). The query above yields two results as there is no error between the query term "foo" and the text node "foo bar", and one error between "foo" and "foa bar".

Functions

Some additional Full-Text Functions have been added to BaseX to extend the official language recommendation with useful features, such as explicitly requesting the score value of an item, marking the hits of a full-text request, or directly accessing the full-text index with the default index options.

Error Messages

Along with the Update Facility, a number of new error codes and messages have been added to the specification and BaseX. All errors are listed in the XQuery Errors overview.

References

- [1] <http://www.w3.org/TR/xpath-full-text-10/>
- [2] <http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/GrGaHo09.pdf>
- [3] <http://files.basex.org/maven/org/apache/lucene-stemmers/3.4.0/lucene-stemmers-3.4.0.jar>
- [4] <http://en.sourceforge.jp/projects/igo/releases/>
- [5] <http://dev.w3.org/2007/xpath-full-text-10-test-suite>
- [6] <http://dev.w3.org/cvsweb/~checkout~/2007/xpath-full-text-10-test-suite/TestSuiteStagingArea/TestSources/thesaurus.xsd?rev=1.3;content-type=application%2Fxml>

Full-Text: Japanese

This article is linked from the Full-Text page. It gives some insight into the implementation of the full-text features for Japanese text corpora. The Japanese version is also available as PDF ^[1]. Thank you to Toshio HIRAI ^[2] for integrating the lexer in BaseX!

Introduction

The lexical analysis of Japanese documents is performed by Igo ^[3]. Igo is a *morphological analyser*, and some of the advantages and reasons for using Igo are:

- compatible with the results of a prominent morphological analyzer "MeCab"
- it can use the dictionary distributed by the Project MeCab
- the morphological analyzer is implemented in Java and is relatively fast

Japanese tokenization will be activated in BaseX if Igo is found in the classpath. igo-0.4.3.jar ^[4] of Igo is currently included in all distributions of BaseX.

In addition to the library, one of the following dictionary files must either be unzipped into the current directory, or into the `etc` sub-directory of the project's Home Directory:

- IPA Dictionary: <http://files.basex.org/etc/ipadic.zip>
- NAIST Dictionary: <http://files.basex.org/etc/naistdic.zip>

Lexical Analysis

The example sentence "私本書(I wrote a book.)" is analyzed as follows.

```
私本書
私 名詞,代名詞,一般,*,*,* 私, , ,
 助詞,係助詞,*,*,*,* , ,
本 名詞,一般,*,*,*,* 本, ,
 助詞,格助詞,一般,*,*,* , ,
書 動詞,自立,*,* 五段行音便,連用形,書, ,
 助動詞,*,*,* 特殊,連用形, , ,
 助動詞,*,*,* 特殊,基本形, , ,
 記号,句点,*,*,*,* , ,
```

The element of the decomposed part is called "Surface", the content analysis is called "Morpheme". The Morpheme component is built as follows:

```
品詞,品詞細分類1,品詞細分類2,品詞細分類3,活用形,活用型,原形,読,発音
(POS, subtyping POS 1, subtyping POS 2, subtyping POS 3, inflections, use type, prototype, reading, pronunciation)
```

Of these, the surface is used as a token. Also, The contents of analysis of a morpheme are used in indexing and stemming.

Parsing

During indexing and parsing, the input strings are split into single *tokens*. In order to reduce the index size and speed up search, the following word classes have been intentionally excluded:

- Mark
- Filler
- Postpositional particle
- Auxiliary verb

Thus, in the example above, the "私", "本", "書" will be passed to the indexer for each token.

Token Processing

"Fullwidth" and "Halfwidth" (which is defined by East Asian Width Properties ^[4]) are not distinguished (this is the so-called ZENKAKU/HANKAKU problem). For example, 書 and 書 will be treated as the same word. If documents are *hybrid*, i.e. written in multiple languages, this is also helpful for some other options of the XQuery Full Text Specification, such as the Case ^[5] or the Diacritics ^[6] Option.

Stemming

Stemming in Japanese means to analyze the results of morphological analysis ("verbs" and "adjectives") that are processed using the "prototype".

If the stemming option is enabled, for example, the two statements "私本書 (I wrote the book)" and "私本書 (I write the book)" can be led back to the same prototype by analyzing their verb:

```
書 動詞, 自立, *, *, 五段行音便, 基本形, [書], 〇, 〇
```

```
書 動詞, 自立, *, *, 五段行音便, 連用接続, [書], 〇, 〇
 助動詞, *, *, *, 特殊, 基本形, 〇, 〇, 〇
```

Because the "auxiliary verb" is always excluded from the tokens, there is no need to consider its use. Therefore, the same result (`true`) is returned for the following two types of queries:

```
'私本書' contains text '書' using stemming using language 'ja'
'私本書' contains text '書' using stemming using language 'ja'
```

Wildcards

The Wildcard option in XQuery Full-Text is available for Japanese as well. The following example is based on '芥川龍之介(AKUTAGAWA, Ryunosuke)', a prominent Japanese writer, the first name of whom is often spelled as "竜之介". The following two queries both return `true`:

```
'芥川龍之介' contains text '.之介' using wildcards using language 'ja'
'芥川竜之介' contains text '.之介' using wildcards using language 'ja'
```

However, there is a special case that requires attention. The following query will yield `false`:

```
'芥川龍之介' contains text '芥川.之介' using wildcards using language 'ja'
```

This is because the next word boundary metacharacters cannot be determined in the query. In this case, you may insert an additional whitespaces as word boundary:

```
'芥川龍之介' contains text '芥川 .之介' using wildcards using language 'ja'
```

As an alternative, you may modify the query as follows:

```
'芥川龍之介' contains text '芥川' ftand '.之介' using wildcards using language 'ja'
```

References

- [1] <http://files.basex.org/etc/ja-ft.pdf>
- [2] <http://blog.infinite.jp>
- [3] <http://igo.sourceforge.jp/>
- [4] <http://unicode.org/Public/UNIDATA/EastAsianWidth.txt>
- [5] <http://www.w3.org/TR/xpath-full-text-10/#ftcaseoption>
- [6] <http://www.w3.org/TR/xpath-full-text-10/#ftdiacriticsoption>

XQuery Update

This article is part of the XQuery Portal. It summarizes the update features of BaseX.

BaseX offers a complete implementation of the XQuery Update Facility (XQUF) ^[1]. This article aims to provide a very quick and basic introduction to the XQUF. First, some examples for update expressions are given. After that, a few problems are addressed that frequently arise due to the nature of the language. These are stated in the Concepts paragraph.

New Functionality

Updating Expressions

There are five new expressions to modify data. While `insert`, `delete`, `rename` and `replace` are basically self-explanatory, the `transform` expression is different, as modified nodes are copied in advance and the original databases remain untouched.

An expression consists of a target node (the node we want to alter) and additional information like insertion nodes, a QName, etc. which depends on the type of expression. Optional modifiers are available for some of them. You can find a few examples and additional information below.

insert

```
insert node (attribute { 'a' } { 5 }, 'text', <e/>) into /n
```

Insert enables you to insert a sequence of nodes into a single target node. Several modifiers are available to specify the exact insert location: insert into **as first/as last**, insert **before/after** and insert **into**.

Note: in most cases, **as last** and **after** will be evaluated faster than **as first** and **before**!

delete

```
delete node //node
```

The example query deletes all `<node>` elements in your database. Note that, in contrast to other updating expressions, the delete expression allows multiple nodes as a target.

replace

```
replace node /n with <a/>
```

The target element is replaced by the DOM node <a/>. You can also replace the value of a node or its descendants by using the modifier **value of**.

```
replace value of node /n with 'newValue'
```

All descendants of /n are deleted and the given text is inserted as the only child. Note that the result of the insert sequence is either a single text node or an empty sequence. If the insert sequence is empty, all descendants of the target are deleted. Consequently, replacing the value of a node leaves the target with either a single text node or no descendants at all.

rename

```
for $n in //node
return rename node $n as 'renamedNode'
```

All node elements are renamed. An iterative approach helps to modify multiple nodes within a single statement. Nodes on the descendant- or attribute-axis of the target are not affected. This has to be done explicitly as well.

Non-Updating Expressions**transform**

```
copy $c := doc('example.xml')//node[@id = 1]
modify rename node $c as 'copyOfNode'
return $c
```

The node element with @id=1 is copied and subsequently assigned a new QName using the rename expression. Note that the transform expression is the only expression which returns an actual XDM instance as a result. You can therefore use it to modify results and especially DOM nodes. This is an issue beginners are often confronted with. More on this topic can be found in the XQUF Concepts section.

The following example demonstrates a common use case:

Query:

```
copy $c :=
  <entry>
    <title>Transform expression example</title>
    <author>BaseX Team</author>
  </entry>
modify (
  replace value of node $c/author with 'BaseX',
  replace value of node $c/title with concat('Copy of: ', $c/title),
  insert node <author>Joey</author> into $c
)
return $c
```

Result:

```
<entry>
  <text>Copy of: Transform expression example</text>
  <author>BaseX</author>
```

```
<author>Joey</author>
</entry>
```

The `<entry>` element (here it is passed to the expression as a DOM node) can also be replaced by a database node, e.g.:

```
copy $c := (db:open('example')//entry)[1]
...
```

In this case, the original database node remains untouched as well, as all updates are performed on the node copy.

Functions

`fn:put()`

`fn:put()` is also part of the XQUF and enables the user to serialize XDM instances to secondary storage. It is executed at the end of a snapshot. Serialized documents therefore reflect all changes made effective during a query.

Some more BaseX-specific updating database functions exist to add, delete or update documents in a database.

XQUF Concepts

There are a few specialties around XQuery Update that you should know about. In addition to the **simple expression**, the XQUF adds the **updating expression** as a new type of expression. An updating expression returns only a Pending Update List (PUL) as a result which is subsequently applied to addressed databases and DOM nodes. A simple expression cannot perform any permanent changes and returns an empty or non-empty sequence.

Pending Update List

The most important thing to keep in mind when using XQuery Update is the Pending Update List (PUL). Updating statements are not executed immediately, but are first collected as update primitives within a set-like structure, the PUL. At the end of a query, all update primitives on this list are applied in a strict order after some compatibility tests. If a conflict exists, an error message is returned and all accessed databases remain untouched (atomicity). For the user this means updates are only visible **after** the end of a snapshot.

If we have the document:

```
<doc> <a/> </doc>
```

... and perform the following query:

```
insert node <b/> into /doc,
for $n in /doc/child::node()
return rename node $n as 'justRenamed'
```

The document looks like this:

```
<doc> <justRenamed/><b/> </doc>
```

Despite explicitly renaming all child nodes of `<doc/>`, the former `<a/>` element is the only one to be renamed. The element is inserted within the same snapshot and is therefore not yet visible to the user.

Returning Results

It is not possible to mix different types of expressions in a query result. The outermost expression of a query must either be a collection of updating or non-updating expressions. There is no way to perform any updating queries and return a result at the same time, except for using the BaseX-specific `db:output()` function, which caches the results of its arguments at runtime and returns it after all updates have been processed.

Example: Perform update and return success message.

```
db:output("Update successful."), insert node <c/> into doc('factbook')/mondial
```

Trying to modify and return a DOM node within the same snapshot is another story. As changes on DOM nodes are non-persistent, you cannot access them in a subsequent query. This is where the transform expression comes into play.

Function Declaration

To use updating expressions within a function, the 'updating' flag has to be added to the function declaration. A correct declaration of a function that contains updating expressions (or one that calls updating functions) looks like this:

```
declare updating function { ... }
```

Effects on your documents

In BaseX, all updates are performed on database nodes. Update operations thus never affect the original input file. You can, however, use the `EXPORT` command or the `fn:put()` function to create an updated XML file. If the database option `WRITEBACK` is turned on, changes in your database are propagated back to the original input file. Make sure you back up your data in advance, as this approach modifies the underlying XML file.

Indexes

As BaseX mainly aims for efficiency, the maintenance of indexes is left to the user. This requires the user to call the `OPTIMIZE` command after all update operations have been executed: multiple update operations can be performed much faster this way, and the database meta data is only updated and regenerated once in the updating process.

Since [Version 7.1](#), incremental index updates are supported for text nodes and attribute values. The `UPDINDEX` database option can be turned on for a particular database to trigger index updates.

Error Messages

Along with the Update Facility, a number of new error codes and messages have been added to the specification and BaseX. All errors are listed in the [XQuery Errors overview](#).

References

[1] <http://www.w3.org/TR/xquery-update-10/>

Serialization

This page is part of the XQuery Portal. Serialization parameters define how XQuery items and XML nodes are textually output, i.e., *serialized*. (For input see Parsers.) They have been formalized in the W3C XQuery Serialization 3.0 ^[1] document. In BaseX, they can be specified in several ways:

- by including them in the prolog of the XQuery expression,
- by specifying them in the XQuery functions `file:write()` or `fn:serialize()`,
- by using the `-s` flag of the BaseX command-line clients,
- by setting the `SERIALIZER` option before running a query,
- by setting the `EXPORTER` option before exporting a database, or
- by setting them as REST query parameters

Parameters

The following table gives a brief summary of all serialization parameters recognized by BaseX. For details, please refer to official specification.

Parameter	Description	Allowed	Default	Examples
<code>method</code>	Specifies the serialization method: <ul style="list-style-type: none"> • <code>xml</code>, <code>xhtml</code>, <code>html</code>, and <code>text</code> are adopted from the official specification. • <code>html5</code> is specific to BaseX and can be used to output query results as HTML5 ^[2]. • <code>json</code> and <code>jsonml</code> are specific to BaseX and can be used to output XML nodes in the JSON format (see the JSON Module for more details). • <code>raw</code> is BaseX-specific as well: Binary data types are output in their <i>raw</i> form, i.e., without modifications. For all other types, the items' string values are returned. No indentation takes place, and no characters are encoded via entities. 	<code>xml</code> , <code>xhtml</code> , <code>html</code> , <code>text</code> , <code>html5</code> , <code>json</code> , <code>jsonml</code> , <code>raw</code>	<code>xml</code>	<code>method=xml</code>
<code>version</code>	Specifies the version of the serializer.	<code>xml/xhtml</code> : <code>1.0</code> , <code>1.1</code> <code>html</code> : <code>4.0</code> , <code>4.01</code>	<code>1.0</code>	<code>version=1.0</code>
<code>encoding</code>	Encoding to be used for outputting the data.	<i>all encodings supported by Java</i>	UTF-8	<code>encoding=US-ASCII</code>
<code>indent</code>	Adjusts whitespaces to make the output better readable.	<code>yes</code> , <code>no</code>	<code>yes</code>	<code>indent=no</code>
<code>cdata-section-elements</code>	List of elements to be output as CDATA, separated by whitespaces. Example: <code><text><![CDATA[< >]]></text></code>			<code>cdata-section-elements=text</code>
<code>omit-xml-declaration</code>	Omits the XML declaration, which is serialized before the actual query result Example: <code><?xml version="1.0" encoding="UTF-8"?></code>	<code>yes</code> , <code>no</code>	<code>yes</code>	<code>omit-xml-declaration=no</code>
<code>standalone</code>	Prints or omits the "standalone" attribute in the XML declaration.	<code>yes</code> , <code>no</code> , <code>omit</code>	<code>omit</code>	<code>standalone=yes</code>

doctype-system	Introduces the output with a document type declaration and the given system identifier. Example: <!DOCTYPE x SYSTEM "entities.dtd">			doctype-system=entities.dtd
doctype-public	If doctype-system is specified, adds a public identifier. Example: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">			doctype-public=-//W3C//DTD HTML 4.01//EN, doctype-system=[3]
undeclare-prefixes	Undeclares prefixes in XML 1.1.	yes, no	no	undeclare-prefixes=yes
normalization-form	Specifies a normalization form. BaseX supports Form C (NFC).	NFC, none	NFC	normalization-form=none
media-type	Specifies the media type.		application/xml	media-type=text/plain
use-character-maps	Defines character mappings (not supported).			
byte-order-mark	Prints a byte-order-mark before starting serialization.	yes, no	no	byte-order-mark=yes
escape-uri-attributes	Escapes URI information in certain HTML attributes Example: äöü<a>	yes, no	no	escape-uri-attributes=yes, method=html
include-content-type	Includes a meta content-type element if the result is output as HTML Example: <head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></head>	yes, no	no	include-content-type=yes, method=html

BaseX provides some additional, implementation-specific serialization parameters:

Parameter	Description	Allowed	Default	Examples
format	Turns output formatting on/off, including the conversion of special characters to entities and insertion of item separators. This flag can be turned off to simplify the handling of plain-text results.	yes, no	yes	format=no
tabulator	Uses tab characters (\t) for indenting elements.	yes, no	no	tabulator=yes
indents	Specifies the number of characters to be indented.	positive number	2	indents=1, tabulator=yes
wrap-prefix, wrap-uri	Specifies a prefix and/or URI for wrapping the query results.			wrap-prefix=rest, wrap-uri=http://basex.org/rest
newline	Specifies the type of newline to be used as end-of-line marker.	\n, \r\n, \r	system dependent	newline=\r\n
separator	Determines the string to be used as item separator. Introduced with Version 7.2	arbitrary strings	single space	separator=\n

Changelog

Version 7.2

- Added: `separator` parameter

Version 7.1

- Added: `newline` parameter

Version 7.0

- Added: Serialization parameters added to REST API; JSON/JsonML/raw methods

References

[1] <http://www.w3.org/TR/xslt-xquery-serialization-30>

[2] <http://en.wikipedia.org/wiki/Html5>

[3] <http://www.w3.org/TR/html4/strict.dtd>

XQuery Errors

This article is part of the XQuery Portal. It summarizes all error codes that may be thrown by the BaseX XQuery processor.

As the original specifications are rather bulky and meticulous, we tried our best to make this overview comprehensible to a wider range of readers. The following tables list the error codes that are known to BaseX, a short description, and examples of queries raising that errors.

Original definitions of the error codes are found in the XQuery 3.0 ^[4], XQuery 3.0 Functions ^[6], XQuery 1.0 Update ^[1], XQuery 1.0 Full Text ^[1], and EXPath HTTP ^[1] Specifications.

BaseX Errors

Error Codes: `BASX`

Code	Description	Examples
BASX0001	The specified index type is unknown, or not available.	<code>db:info("unknown")</code>
BASX0002	The specified node is not stored in a database.	<code><xml/>/db:text("word")</code>
BASX0003	The specified database was not found or could not be opened.	<code>db:open('unknown')</code>
BASX0004	The database node referenced by a function is out of range.	<code>db:open-pre('database', -1)</code>
BASX0005	The current user has no permissions to execute an expression.	<code>file:delete('file.txt')</code> : <i>Admin rights needed.</i>
BASX0006	The query was timed out. The <code>TIMEOUT</code> option can be modified in the client/server architecture. This error is only raised by the internal BaseX Java function <code>QueryProcess.next(Item)</code> .	
BASX0007	Nodes were expected as query result. This error is only raised by the internal BaseX Java function <code>QueryProcessor.queryNodes()</code> .	
BASX0008	A database addressed with the <code>doc()</code> function contains more than one document.	<code>doc('collection')</code>
BASX0009	The specified event is unknown.	<code>db:event('unknown', 'event')</code>

BASX0010	The specified database option is unknown.	<code>declare option db:xyz "no"; 1</code>
BASX0011	The specified XSLT parameter XSLT parameter is unknown.	<code>xslt:transform('doc.xml', 'input.xslt', '<INVALID/>')</code>
BASX0011	The specified XSLT parameter XSLT parameter is unknown.	<code>xslt:transform('doc.xml', 'input.xslt', '<INVALID/>')</code>
BASX0012	A single document is expected as replace target.	<code>db:replace('db', 'unknown.xml', '<a/>')</code>
BASX0013	An empty rename sting was specified.	<code>db:rename('db', 'old.xml',)</code>
BASX0014	An error occurred while optimizing the database.	<code>db:optimize('db')</code>
BASX0015	The specified input text cannot be parsed as JSON.	<code>json:parse()</code>
BASX0016	The specified input text cannot be serialized to a JSON document.	<code>json:serialize(<invalid/>)</code>
BASX0017	Invalid values were used for <code>fn:partial-apply()</code> .	
BASX0018	Bytes could not be converted to a string.	

Static Errors

Error Codes: XPST, XQST

Code	Description	Examples
XPST0003	An error occurred while <i>parsing</i> the query string (i.e., before the query could be compiled and executed). This error is the most common one, and may be accompanied by a variety of different error messages.	<code>1+for i in /* return \$i</code>
XPST0005	An expression will never results, no matter what input will be processed.	<code>doc('input')/..</code>
XPST0008	A variable or type name is used that has not been defined in the current scope.	<code>\$a--element(*, x)</code>
XPST0017	<ul style="list-style-type: none"> The specified function is unknown, or it uses the wrong number of arguments. 	<code>unknown() count(1,2,3)</code>
XPST0051	An unknown QName is used in a <i>sequence type</i> (e.g. in the target type of the <code>cast</code> expression).	<code>1 instance of x"test" cast as xs:ittr</code>
XPST0080	<code>xs:NOTATION</code> or <code>xs:anyAtomicType</code> is used as target type of <code>cast</code> or <code>castable</code> .	<code>1 castable as xs:NOTATION</code>
XPST0081	<ul style="list-style-type: none"> A QName uses a prefix that has not been bound to any namespace, or a pragma or option declaration has not been prefixed. 	<code>unknown:x(# pragma #) { 1 }</code>
XQST0009	The query imports a schema (schema import is not supported by BaseX).	<code>import schema "x"; ()</code>
XQST0022	Namespace values must be constant strings.	<code><elem xmlns="{ 'dynamic' }"/></code>
XQST0031	The specified XQuery version is not specified.	<code>xquery version "9.9"; ()</code>
XQST0032	The base URI was declared more than once.	<code>declare base-uri ...</code>
XQST0033	A namespace prefix was declared more than once.	<code>declare namespace a="a"; declare namespace a="b"; ()</code>
XQST0034	A function was declared more than once.	<code>declare function local:a() { 1 }; declare function local:a() { 2 }; local:a()</code>
XQST0038	The default collation was declared more than once.	<code>declare default collation ...</code>

XQST0039	Two or more parameters in a user-defined function have the same name.	<pre>declare function local:fun(\$a, \$a) { \$a * \$a }; local:fun(1,2)</pre>
XQDY0040	Two or more attributes in an element have the same node name.	<pre><elem a="1" a="12"/></pre>
XQDY0045	A user-defined function uses a reserved namespace.	<pre>declare function fn:fun() { 1 }; ()</pre>
XQST0047	A module was defined more than once.	<pre>import module ...</pre>
XQST0048	A module declaration does not match the namespace of the specified module.	<pre>import module namespace invalid="uri"; 1</pre>
XQST0049	A global variable was declared more than once.	<pre>declare variable \$a := 1; declare variable \$a := 1; \$a</pre>
XQST0054	A global variable depends on itself. This may be triggered by a circular variable definition.	<pre>declare variable \$a := local:a(); declare function local:a() { \$a }; \$a</pre>
XQST0055	The mode for copying namespaces was declared more than once.	<pre>declare copy-namespaces ...</pre>
XQST0057	The namespace of a schema import may not be empty.	<pre>import schema ""; ()</pre>
XQST0059	The schema or module with the specified namespace cannot be found or processed.	<pre>import module "unknown"; ()</pre>
XQST0060	A user-defined function has no namespace.	<pre>declare default function namespace ""; declare function x() { 1 }; 1</pre>
XQST0065	The ordering mode was declared more than once.	<pre>declare ordering ...</pre>
XQST0065	The default namespace mode for elements or functions was declared more than once.	<pre>declare default element namespace ...</pre>
XQST0067	The construction mode was declared more than once.	<pre>declare construction ...</pre>
XQST0068	The mode for handling boundary spaces was declared more than once.	<pre>declare boundary-space ...</pre>
XQST0069	The default order for empty sequences was declared more than once.	<pre>declare default order empty ...</pre>
XQST0070	A namespace declaration overwrites a reserved namespace.	<pre>declare namespace xml=""; ()</pre>
XQST0071	A namespace is declared more than once in an element constructor.	<pre></pre>
XQST0075	The query contains a validate expression (validation is not supported by BaseX).	<pre>validate strict { () }</pre>
XQST0076	A <code>group by</code> or <code>order by</code> clause specifies an unknown collation.	<pre>for \$i in 1 to 10 order by \$i collation "unknown" return \$i</pre>
XQST0079	A pragma was specified without the expression that is to be evaluated.	<pre>(# xml:a #) {}</pre>
XQST0085	An empty namespace URI was specified.	<pre><pref:elem xmlns:pref=""/></pre>
XQST0087	An unknown encoding was specified. Note that the encoding declaration is currently ignored in BaseX.	<pre>xquery version "1.0" encoding "a b"; ()</pre>
XQST0088	An empty module namespace was specified.	<pre>import module ""; ()</pre>
XQST0089	Two variables in a <code>for</code> or <code>let</code> clause have the same name.	<pre>for \$a at \$a in 1 return \$i</pre>
XQST0090	A character reference specifies an invalid character.	<pre>" "</pre>
XQST0093	A module depends on itself. This may be triggered by a circular module definition.	<pre>import module ...</pre>
XQST0094	<code>group by</code> references a variable that has not been declared before.	<pre>for \$a in 1 group by \$b return \$a</pre>

XQST0097	A <code>decimal-format</code> property is invalid.	<code>declare default decimal-format digit = "xxx"; 1</code>
XQST0098	A single <code>decimal-format</code> character was assigned to multiple properties.	<code>declare default decimal-format digit = "%"; 1</code>
XQST0099	The context item was declared more than once.	<code>declare context item ...</code>
XQST0107	The initializer of the context item depends on itself.	<code>declare context item := .; ()</code>
XQST0108	Output declarations may only be specified in the main module.	Module: <code>declare output ...</code>
XQST0109	The specified serialization parameter is unknown.	<code>declare option output:unknown "..."; 1</code>
XQST0110	A serialization parameter was specified more than once in the output declarations.	<code>declare option output:indent "no"; declare option output:indent "no"; 1</code>
XQST0111	A decimal format was declared more than once.	<code>declare decimal-format ...</code>
XQST0113	Context item values may only be in the main module.	Module: <code>declare context item := 1;</code>
XQST0114	A <code>decimal-format</code> property has been specified more than once.	<code>declare decimal-format EN NaN="!" NaN="?"; ()</code>

Type Errors

Error Codes: XPTY, XQTY

Code	Description	Examples
XPTY0004	This error is raised if an expression has the wrong type, or cannot be cast into the specified type. It may be raised both statically (during query compilation) or dynamically (at runtime).	<code>1 + "A"abs("a")1</code> cast as <code>xs:gYear</code>
XPTY0018	The result of the last step in a path expression contains both nodes and atomic values.	<code>doc('input.xml')/(*, 1)</code>
XPTY0019	The result of a step (other than the last step) in a path expression contains an atomic values.	<code>(1 to 10)/*</code>
XQTY0024	An attribute node cannot be bound to its parent element, as other nodes of a different type were specified before.	<code><elem>text { attribute a { "val" } }</elem></code>

Dynamic Errors

Error Codes: XPDY, XQDY

Code	Description	Examples
XPDY0002	<ul style="list-style-type: none"> No value has been defined for an external variable, or no context item has been set before the query was executed. 	<pre>declare variable \$x external; \$xdescendant::*</pre>
XPDY0050	<ul style="list-style-type: none"> The operand type of a <code>treat</code> expression does not match the type of the argument, or the root of the context item must be a document node. 	<pre>"string" treat as xs:int"string"[/]</pre>
XQDY0025	Two or more attributes in a constructed element have the same node name.	<pre>element x { attribute a { "" } attribute a { "" } }</pre>
XQDY0026	The content of a computed processing instruction contains "?>".	<pre>processing-instruction pi { "?>" }</pre>
XQDY0041	The name of a processing instruction is invalid.	<pre>processing-instruction { "1" } { "" }</pre>
XQDY0043	The node name of an attribute uses reserved prefixes or namespaces.	<pre>attribute xmlns { "etc" }</pre>
XQDY0064	The name of a processing instruction equals "XML" (case insensitive).	<pre>processing-instruction xml { "etc" }</pre>
XQDY0072	The content of a computed comment contains "--" or ends with "-".	<pre>comment { "one -- two" }</pre>
XQDY0074	The name of a computed attribute or element is invalid, or uses an unbound prefix.	<pre>element { "x y" } { "" }</pre>
XQDY0095	A sequence with more than one item was bound to a <code>group by</code> clause.	<pre>let \$a := (1,2) group by \$a return \$a</pre>
XQDY0096	The node name of an element uses reserved prefixes or namespaces.	<pre>element { QName("uri", "xml:n") } {}</pre>

Functions Errors

Error Codes: FOAR, FOCA, FOCH, FODC, FODF, FODT, FOER, FOFD, FONS, FORG, FORX, FOTY, FOUT

Code	Description	Examples
FOAR0001	A value was divided by zero.	<pre>1 div 0</pre>
FOAR0002	A numeric declaration or operation causes an over- or underflow.	<pre>12345678901234567890xs:double("-INF") idiv 1</pre>
FOCA0002	<ul style="list-style-type: none"> A float number cannot be converted to a decimal or integer value, or a function argument cannot be converted to a valid QName. 	<pre>xs:int(xs:double("INF"))QName("", "el em")</pre>
FOCA0003	A value is too large to be represented as integer.	<pre>xs:integer(99e100)</pre>
FOCA0005	"NaN" is supplied to duration operations.	<pre>xs:yearMonthDuration("PLY") * xs:double("NaN")</pre>
FOCH0001	A codepoint was specified that does not represent a valid XML character.	<pre>codepoints-to-string(0)</pre>
FOCH0002	A unsupported collation was specified in a function.	<pre>compare('a', 'a', 'unknown')</pre>
FOCH0003	A unsupported normalization form was specified in a function.	<pre>normalize-unicode('a', 'unknown')</pre>
FODC0001	The argument specified in <code>fn:id()</code> or <code>fn:idref()</code> must have a document node as root.	<pre>id("id0", <xml/>)</pre>
FODC0002	The specified document resource cannot be retrieved.	<pre>doc("unknown.xml")</pre>

FODC0004	The specified collection cannot be retrieved.	<code>collection("unknown")</code>
FODC0005	The specified URI to a document resource is invalid.	<code>doc("<xml/>")</code>
FODC0006	The string passed to <code>fn:parse-xml()</code> is not well-formed.	<code>parse-xml("<x/")</code>
FODC0007	The base URI passed to <code>fn:parse-xml()</code> is invalid.	<code>parse-xml("<x/>", ":")</code>
FODF1280	The name of the decimal format passed to <code>fn:format-number()</code> is invalid.	<code>format-number(1, "0", "invalid")</code>
FODF1310	The picture string passed to <code>fn:format-number()</code> is invalid.	<code>format-number(1, "invalid")</code>
FODT0002	A duration declaration or operation causes an over- or underflow.	<code>implicit-timezone() div 0</code>
FODT0003	An invalid timezone was specified.	<code>adjust-time-to-timezone(xs:time("01:01:01"), xs:dayTimeDuration("PT20H"))</code>
FOER0000	Error triggered by the <code>fn:error()</code> function.	<code>error()</code>
FOFD1340	The picture string passed to <code>fn:format-date()</code> , <code>fn:format-time()</code> or <code>fn:format-dateTime()</code> is invalid.	<code>format-date(current-date(), "[]")</code>
FOFD1350	The picture string passed to <code>fn:format-date()</code> , <code>fn:format-time()</code> or <code>fn:format-dateTime()</code> specifies a non-available component.	<code>format-time(current-time(), "[Y2]")</code>
FONS0004	A function has a QName as argument that specifies an unbound prefix.	<code>resolve-QName("x:e", <e/>)</code>
FORG0001	A value cannot be cast to the required target type.	<code>xs:integer("A")1 + <x>a</x></code>
FORG0002	The URI passed to <code>fn:resolve-URI()</code> is invalid.	<code>resolve-URI(":")</code>
FORG0003	<code>fn:zero-or-one()</code> was called with more than one item.	<code>zero-or-one((1, 2))</code>
FORG0004	<code>fn:one-or-more()</code> was called with zero items.	<code>one-or-more()</code>
FORG0005	<code>fn:exactly-one()</code> was called with zero or more than one item.	<code>exactly-one((1, 2))</code>
FORG0006	A wrong argument type was specified in a function call.	<code>sum(1, "string")</code>
FORG0008	The arguments passed to <code>fn:dateTime()</code> have different timezones.	<code>dateTime(xs:date("2001-01-01+01:01"), current-time())</code>
FORX0001	A function specifies an invalid regular expression flag.	<code>matches('input', 'query', 'invalid')</code>
FORX0002	A function specifies an invalid regular expression.	<code>matches('input', '[')</code>
FORX0003	A regular expression matches an empty string.	<code>tokenize('input', '?.?')</code>
FORX0004	The replacement string of a regular expression is invalid.	<code>replace("input", "match", "\")</code>
FOTY0012	An item has no typed value.	
FOTY0013	Functions items cannot be atomized, have no defined equality, and have no string representation.	
FOTY0014	Function items have no string representation.	
FOTY0015	Function items cannot be compared.	

FOUT1170	Function argument cannot be used to retrieve a text resource.	
FOUT1170	Encoding to retrieve a text resource is invalid or not supported.	<code>unparsed-text('file.txt', 'InvalidEncoding')</code>

....to be added: FOTY0012-0015

Serialization Errors

Error Codes: SEPM, SERE, SESU

Code	Description	Examples
SESU0007	The specified encoding is not supported.	<code>declare option output:encoding "xyz"; 1</code>
SEPM0009	<code>omit-xml-declaration</code> is set to <code>yes</code> , and <code>standalone</code> has a value other than <code>omit</code> .	
SEPM0010	<code>method</code> is set to <code>xml</code> , <code>undeclare-prefixes</code> is set to <code>yes</code> , and <code>version</code> is set to <code>1.0</code> .	
SERE0014	<code>method</code> is set to <code>html</code> , and an invalid HTML character is found.	
SERE0015	<code>method</code> is set to <code>html</code> , and a closing bracket (>) appears inside a processing instruction.	
SEPM0016	A specified parameter is unknown or has an invalid value.	<code>declare option output:indent "nope"; 1</code>
SEPM0017	The definition of serialization parameter is invalid.	

Update Errors

Error Codes: FOUP, XUDY, XUST, XUTY

Code	Description	Examples
FOUP0001	The first argument of <code>fn:put()</code> must be a document node or element.	<code>fn:put(text { 1 }, 'file.txt')</code>
FOUP0002	The second argument of <code>fn:put()</code> is not a valid URI.	<code>fn:put(<a/>, '//')</code>
XUDY0009	The target node of a replace expression needs a parent in order to be replaced.	<code>replace node <target/> with <new/></code>
XUDY0014	The expression updated by the <code>modify</code> clause was not created by the <code>copy</code> clause.	<code>let \$a := doc('a') return copy \$b := \$a modify delete node \$a/* return \$b</code>
XUDY0015	In a <code>rename</code> expression, a target is renamed more than once.	<code>let \$a := <xml/> return (rename node \$a as 'a', rename node \$a as 'b')</code>
XUDY0016	In a <code>replace</code> expression, a target is replaced more than once.	<code>let \$a := <x>x</x>/node() return (replace node \$a with <a/>, replace node \$a with)</code>
XUDY0017	In a <code>replace value of</code> expression, a target is replaced more than once.	<code>let \$a := <x/> return (replace value of node \$a with 'a', replace value of node \$a with 'a')</code>
XUDY0021	The resulting update expression contains duplicate attributes.	<code>copy \$c := <x a='a'/> modify insert node attribute a {""} into \$c return \$c</code>
XUDY0023	The resulting update expression conflicts with existing namespaces.	<code>rename node <a:ns xmlns:a='uri'/> as QName('URI', 'a:ns')</code>

XUDY0024	New namespaces conflict with each other.	<code>copy \$n := <x/> modify (insert node attribute { QName('uri1', 'a') } { "" } into \$n, insert node attribute { QName('uri2', 'a') } { "" } into \$n) return \$n</code>
XUDY0027	Target of an update expression is an empty sequence.	<code>insert node <x/> into ()</code>
XUDY0029	The target of an update expression has no parent node.	<code>insert node <new/> before <target/></code>
XUDY0030	Attributes cannot be inserted before or after the child of a document node.	<code>insert node <e a='a' /> /@a after document { <e/> } /*</code>
XUDY0031	Multiple calls to <code>fn:put()</code> address the same URI.	<code>for \$i in 1 to 3 return put(<a/>, 'file.txt')</code>
XUST0001	No updating expression is allowed here.	<code>delete node /, "finished."</code>
XUST0002	An updating expression is expected in the <code>modify</code> clause or an updating function.	<code>copy \$a := <x/> modify 1 return \$a</code>
XUST0003	The revalidation mode was declared more than once.	<code>declare revalidation ...</code>
XUST0026	The query contains a revalidate expression (revalidation is not supported by BaseX).	<code>declare revalidation ...</code>
XUST0028	no return type may be specified in an updating function.	<code>declare updating function local:x() as item() { () }; ()</code>
XUTY0004	New attributes to be inserted must directly follow the root node.	<code>insert node (<a/>, attribute a { "" }) into <a/></code>
XUTY0005	A single element or document node is expected as target of an <code>insert</code> expression.	<code>insert node <new/> into attribute a { "" }</code>
XUTY0006	A single element, text, comment or processing instruction is expected as target of an <code>insert before/after</code> expression.	<code>insert node <new/> after attribute a { "" }</code>
XUTY0007	Only nodes can be deleted.	<code>delete node "string"</code>
XUTY0008	A single element, text, attribute, comment or processing instruction is expected as target of a <code>replace</code> expression.	<code>replace node document { <a/> } with</code>
XUTY0010	In a <code>replace</code> expression, in which no attributes are targeted, the replacing nodes must not be attributes as well.	<code>replace node <a>/b with attribute size { 1 }</code>
XUTY0011	In the <code>replace</code> expression, in which attributes are targeted, the replacing nodes must be attributes as well.	<code>replace node <e a="" /> /@a with <a/></code>
XUTY0012	In a <code>rename</code> expression, the target nodes must be an element, attribute or processing instruction.	<code>rename node text { 1 } as <x/></code>
XUTY0013	An expression in the <code>copy</code> clause must return a single node.	<code>copy \$c := (<a/>,) modify () return \$c</code>
XUTY0022	An attribute must not be inserted into a document node.	<code>insert node <e a="" /> /@a into document { 'a' }</code>

Full-Text Errors

Error Codes: FTDY, FTST

Code	Description	Examples
FTDY0016	The specified weight value is out of range.	'a' contains text 'a' weight { 1001 }
FTDY0017	The <code>not in</code> operator contains a <i>string exclude</i> .	'a' contains text 'a' not in (ftnot 'a')
FTDY0020	The search term uses an invalid wildcard syntax.	'a' contains text '{.}' using wildcards
FTST0000	BaseX specific: Either wildcard or the fuzzy option can be chosen at the same time.	'a' contains text 'a' using wildcards using fuzzy
FTST0007	The full-text expression contains an ignore option (the <code>ignore option</code> is not supported by BaseX).	'a' contains text 'a' without content 'x'
FTST0008	The specified stop word file could not be opened or processed.	'a' contains text 'a' using stop words at 'unknown.txt'
FTST0009	The specified language is not supported.	'a' contains text 'a' using language 'aaa'
FTST0018	The specified thesaurus file could not be opened or processed.	'a' contains text 'a' using thesaurus at 'aaa'
FTST0019	A match option was defined more than once.	'a' contains text 'a' using stemming using stemming

ZIP Module Errors

Error Codes: FOZP

Code	Description	Examples
FOZP0001	The specified path does not exist.	zip:entries('unknown.zip')
FOZP0002	Entries in the description of a ZIP archive are unknown, missing, or invalid.	zip:zip-file(<file xmlns="unknown" href='target.zip'/>)
FOZP0003	ZIP file extraction or creation fails for some other reason (e.g.: new ZIP file contains no entries, or duplicates).	

Cryptographic Module Errors

Error Codes: FOCX

Code	Description	Examples
FOCX0001	Canonicalization algorithm is not supported.	
FOCX0002	Digest algorithm is not supported.	
FOCX0003	Signature algorithm is not supported.	
FOCX0004	XPath expression is invalid.	
FOCX0005	Invalid name for \$digital-certificate root.	
FOCX0006	Invalid child element of \$digital-certificate.	
FOCX0007	Key store is null.	
FOCX0008	I/O error while reading keystore.	
FOCX0009	Permission denied to read keystore.	

FOCX0010	Keystore URL is invalid.	
FOCX0011	Keystore type is not supported.	
FOCX0012	Cannot find key for alias in given keystore.	
FOCX0013	Hashing algorithm is not supported.	
FOCX0014	Encoding method is not supported.	
FOCX0015	Cannot find signature element.	
FOCX0016	No such padding.	
FOCX0017	Incorrect padding.	
FOCX0018	Encryption type is not supported.	
FOCX0019	Secret key is invalid.	
FOCX0020	Illegal block size.	
FOCX0021	Algorithm is not supported.	
FOCX0022	Decryption type is not supported.	
FOCX9999	Signature type is not supported.	
FOCX9998	Not (yet) supported.	
FOCX9997	Algorithm not compatible with encryption type.	
FOCX9996	IO Exception.	
FOCX9995	Keystore exception.	
FOCX9994	Signature exception.	
FOCX9993	Invalid algorithm.	
FOCX9992	Invalid certificate alias.	

File Module Errors

Error Codes: FOFL

Code	Description	Examples
FOFL0001	The specified path does not exist.	<code>file:size('unknown.txt')</code>
FOFL0002	The specified path does already exist.	<code>file:create-directory('existing-directory')</code>
FOFL0003	The specified path does not point to a directory.	<code>file:list('file.txt')</code>
FOFL0004	The specified path points to a directory.	<code>file:read('directory')</code>
FOFL0005	The specified encoding is not supported.	<code>file:read('file.txt', 'UTF99')</code>
FOFL9999	A file operation fails for any other reason.	

HTTP Module Errors

Error Codes: FOHC

Code	Description	Examples
FOHC0001	The specified URL is invalid.	
FOHC0002	The requested method is not valid for HTTP.	
FOHC0004	The request element is not valid.	
FOHC0005	An HTTP error occurred.	
FOHC0006	The provided credentials are invalid.	
FOHC0007	The HTML input could not be parsed.	

Packaging Errors

Error Codes: PACK

Code	Description	Examples
PACK0001	The specified package does not exist.	
PACK0002	The package uses an invalid namespace URI.	
PACK0003	A required package is not installed.	
PACK0004	Invalid package descriptor found.	
PACK0005	A module is already installed within another package.	
PACK0006	The current package could not be parsed.	
PACK0007	A package cannot be deleted.	
PACK0008	A package depends on another package.	
PACK0009	The addressed package version is not supported.	
PACK0010	Invalid JAR descriptor found.	
PACK0011	The JAR descriptor could not be read.	

SQL Module Errors

Error Codes: FOSQ

Code	Description	Examples
FOSQ0001	An SQL exception occurred.	
FOSQ0002	No opened connection with the specified id found.	
FOSQ0003	Number of parameters differs from number of placeholders.	
FOSQ0004	No parameter type specified.	
FOSQ0005	An unexpected attribute was found.	
FOSQ0006	The specified format is illegal.	
FOSQ0007	The JDBC driver cannot be initialized.	

References

- [1] <http://www.expath.org/spec/http-client>
-

XQuery Modules

Cryptographic Module

This XQuery Module contains functions to perform cryptographic operations in XQuery. The cryptographic module is based on an early draft of the EXPath Cryptographic Module ^[1] and provides the following functionality:

1. Creation of message authentication codes (HMAC)
2. Encryption and decryption
3. Creation and validation of an XML Digital Signature

All functions are introduced with the `crypto:` prefix, which is linked to the statically declared `http://expath.org/ns/crypto` namespace.

Message Authentication Code

`crypto:hmac`

Signatures `crypto:hmac($message as xs:string(), $secret-key as xs:string(), algorithm as xs:string()) as xs:string()`
`crypto:hmac($message as xs:string(), $secret-key as xs:string(), algorithm as xs:string(), $encoding as xs:string()) as xs:string()`

Summary Creates a message authentication code via a cryptographic hash function and a secret key. `$encoding` must either be `hex`, `base64` or the empty string and specifies the encoding of the returned authentication code. **Default is base64.** `$algorithm` describes the hash algorithm which is used for encryption. Currently supported are `md5`, `sha1`, `sha256`, `sha384`, `sha512`. **Default is md5.**

Errors **FOCX0013** is raised if the specified hashing algorithm is not supported.
FOCX0014 is raised if the specified encoding method is not supported.
FOCX0019 is raised if the specified secret key is invalid.

Example Returns the message authentication code (MAC) for a given string.

Query:

```
crypto:hmac('message', 'secretkey', 'md5', 'base64')
```

Result:

```
34D1E3818B347252A75A4F6D747B21C2
```

Encryption & Decryption

The encryption and decryption functions underlie several limitations:

- Cryptographic algorithms are currently limited to `symmetric` algorithms only. This means that the same secret key is used for encryption and decryption.
- Available algorithms are `DES` and `AES`.
- Padding is fixed to `PKCS5Padding`.
- The result of an encryption using the same message, algorithm and key looks different each time it is executed. This is due to a random initialization vector (IV) which is appended to the message and simply increases security.
- As the IV has to be passed along with the encrypted message somehow, data which has been encrypted by the `crypto:encrypt` function in BaseX can only be decrypted by calling the `crypto:decrypt` function.

crypto:encrypt

Signatures `crypto:encrypt` (\$input as xs:string(), \$encryption-type as xs:string(), \$secret-key as xs:string(), \$cryptographic-algorithm as xs:string()) as xs:string()

Summary Encrypts the given input string.

`$encryption-type` must be `symmetric`, as asymmetric encryption is not supported so far. **Default is `symmetric`**.
`$secret-key` is the secret key which is used for both encryption and decryption of input data. Its length is fixed and depends on the chosen algorithm: 8 bytes for `DES`, 16 bytes for `AES`.
`$cryptographic-algorithm` must either be `DES` or `AES`. Other algorithms are not supported so far, but, of course, can be added on demand. **Default is `DES`**.

Errors **FOCX0016** is raised if padding problems arise.
FOCX0017 is raised if padding is incorrect.
FOCX0018 is raised if the encryption type is not supported.
FOCX0019 is raised if the secret key is invalid.
FOCX0020 is raised if the block size is incorrect.
FOCX0021 is raised if the specified encryption algorithm is not supported.

Example Encrypts input data.

Query:

```
crypto:encrypt('message', 'symmetric', 'keykeyke', 'DES')
```

crypto:decrypt

Signatures `crypto:decrypt` (\$input as xs:string(), \$decryption-type as xs:string(), \$secret-key as xs:string(), \$cryptographic-algorithm as xs:string()) as xs:string()

Summary Decrypts the encrypted \$input.

`$decryption-type` must be `symmetric`. An option for asymmetric encryption will most likely be added with another version of BaseX. **Default is `symmetric`**.
`$secret-key` is the secret key which is used for both encryption and decryption of input data. Its length is fixed and depends on the chosen algorithm: 8 bytes for `DES`, 16 bytes for `AES`.
`$cryptographic-algorithm` must either be `DES` or `AES`. Other algorithms are not supported so far, but, of course, can be added on demand. **Default is `DES`**.

Errors **FOCX0016** is raised if padding problems arise.
FOCX0017 is raised if padding is incorrect.
FOCX0018 is raised if the encryption type is not supported.
FOCX0019 is raised if the secret key is invalid.
FOCX0020 is raised if the block size is incorrect.
FOCX0021 is raised if the specified encryption algorithm is not supported.

Example Decrypts input data and returns the original string.

Query:

```
let $encrypted := crypto:encrypt('message', 'symmetric','keykeyke','DES')
return crypto:decrypt($encrypted, 'symmetric','keykeyke','DES')
```

Result:

```
message
```

XML Signature

XML Signatures ^[1] are used to sign data. In our case, the data which is signed is an XQuery node. The following example shows the basic structure of an XML signature.

XML Signature

```
<Signature>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    <Reference>
      <Transforms/>
      <DigestMethod/>
      <DigestValue/>
    </Reference>
  </SignedInfo>
  <SignatureValue/>
  <KeyInfo/>
  <Object/>
</Signature>
```

- **SignedInfo** contains or references the signed data and lists algorithm information
- **Reference** references the signed node
- **Transforms** contains transformations (i.e. XPath expressions) that are applied to the input node in order to sign a subset
- **DigestValue** holds digest value of the transformed references
- **SignatureValue** contains the Base64 encoded value of the encrypted digest of the SignedInfo element
- **KeyInfo** provides information on the key that is used to validate the signature
- **Object** contains the node which is signed if the signature is of type enveloping

Signature Types

Depending on the signature type, the signature element is either placed as a child of the signed node (enveloped type), or directly contains the signed node (enveloping type). Detached signatures are so far not supported.

Digital Certificate

The generate-signature function allows to pass a digital certificate. This certificate holds parameters that allow to access key information stored in a Java key store which is then used to sign the input document. Passing a digital certificate simply helps re-using the same key pair to sign and validate data. The digital certificate is passed as a node and has the following form:

```
<digital-certificate>
  <keystore-type>JKS</keystore-type>
  <keystore-password>...</keystore-password>
  <key-alias>...</key-alias>
  <private-key-password>...</private-key-password>
  <keystore-uri>...</keystore-uri>
</digital-certificate>
```

crypto:generate-signature

Signatures `crypto:generate-signature`(\$input-doc node(), \$canonicalization-algorithm as xs:string(), \$digest-algorithm as xs:string(), \$signature-algorithm as xs:string(), \$signature-namespace-prefix as xs:string(), \$signature-type as xs:string()) as node()
`crypto:generate-signature`(\$input-doc node(), \$canonicalization-algorithm as xs:string(), \$digest-algorithm as xs:string(), \$signature-algorithm as xs:string(), \$signature-namespace-prefix as xs:string(), \$signature-type as xs:string(), \$xpath-expression as xs:string()) as node()
`crypto:generate-signature`(\$input-doc node(), \$canonicalization-algorithm as xs:string(), \$digest-algorithm as xs:string(), \$signature-algorithm as xs:string(), \$signature-namespace-prefix as xs:string(), \$signature-type as xs:string(), \$digital-certificate as node()) as node()
`crypto:generate-signature`(\$input-doc node(), \$canonicalization-algorithm as xs:string(), \$digest-algorithm as xs:string(), \$signature-algorithm as xs:string(), \$signature-namespace-prefix as xs:string(), \$signature-type as xs:string(), \$xpath-expression as xs:string(), \$digital-certificate as node()) as node()

Summary \$canonicalization-algorithm must either be inclusive-with-comments, inclusive, exclusive-with-comments or exclusive. **Default is inclusive-with-comments.**
 \$digest-algorithm must be one of the following: SHA1, SHA256 or SHA512. **Default is SHA1.**
 \$signature-algorithm must either be RSA_SHA1 or DSA_SHA1. **Default is RSA_SHA1.**
 \$signature-namespace-prefix may be empty and prefixes the Signature element accordingly.
 \$signature-type must either be enveloped or enveloping. Detached signatures are so far not supported. **Default is enveloped.**
 \$xpath-expression is an arbitrary XPath expression which specifies a subset of the document that is to be signed.
 \$digital-certificate is the digital certificate used to sign the input document.

Errors **FOCX0001** is raised if the canonicalization algorithm is not supported.
FOCX0002 is raised if the digest algorithm is not supported.
FOCX0003 is raised if the signature algorithm is not supported.
FOCX0004 is raised if the \$xpath-expression is invalid.
FOCX0005 is raised if the root name of \$digital-certificate is not 'digital-certificate'.
FOCX0007 is raised if the key store is null.
FOCX0012 is raised if the key cannot be found in the specified key store.
FOCX9992 is raised if the certificate alias is invalid.
FOCX9993 is raised if an invalid algorithm is specified.
FOCX9994 is raised if an exception occurs while the signing the document.
FOCX9995 is raised if an exception occurs during key store initialization.
FOCX9996 is raised if an IO exception occurs.
FOCX9999 is raised if the signature type is not supported.

Example Generates an XML Signature ^[1].

Query:

```
crypto:generate-signature(<a/>, '', '', '', '', '')
```

Result:

```

<a>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>9hvH4qztnIYgYfJDRLnEMPJdoaY=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>Pn/Jr44WBcdARff2UVYEiwYW1563XdqnU87nusAIaHgzd+U3SrjVJhPFLDe0DJfxVtYzLFaznTYE
P3ddeoFmyA==</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>rtvpFSbCIE2BJePlVYLIRIjXl0R7ESr2+D+JOVKn7AM7VZbcbRDPeQrBjSkEz1HWC/N067tjB3qH
4/4PPT9bGQ==</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</a>

```

crypto:validate-signature

Signatures `crypto:validate-signature($input-doc as node()) as xs:boolean()`

Summary Checks if the given node contains a `Signature` element and whether the signature is valid. In this case `true` is returned. If the signature is invalid the function returns `false`.

Errors **FOCX0015** is raised if the signature element cannot be found.
FOCX9994 is raised if an unspecified problem occurs during validation.
FOCX9996 is raised if an IO exception occurs during validation.

Example **Validates an XML Signature** ^[1].

Query:

```

let $sig := crypto:generate-signature(<a/>, '', '', '', '', '')
return crypto:validate-signature($sig)

```

Result:

```
true
```

Changelog

The Module was introduced with Version 7.0.

References

[1] <http://www.w3.org/TR/xmldsig-core/>

Database Module

This XQuery Module contains functions for handling databases from within XQuery. Existing databases can be opened and listed, index structures can be directly accessed, documents can be added to and removed from collections, etc. All functions are introduced with the `db:` prefix, which is linked to the statically declared `http://basex.org/modules/db` namespace.

Commonalities

Many functions share `$db` as argument, which may either reference a string, denoting the name of the addressed database, or a node from an already opened database. The following errors may be raised by these functions:

- **BASX0002** is raised if `$db` references a node that is not stored in a database (i.e., references a main-memory XML fragment).
- **BASX0003** is raised if the addressed database cannot be opened.

General Functions

db:info

Signatures `db:info($db as item()) as element(Database)`

Summary Returns meta information on the specified database.

db:list

Signatures `db:list() as xs:string*`

`db:list($db as item()) as xs:string*`

`db:list($db as item(), $path as xs:string) as xs:string*`

Summary Returns an `xs:string` sequence with the names of all databases.

If `$db` is specified, all documents and raw files of the specified database are returned.

The list of resources can be further restricted by the `$path` argument.

Examples • `db:list("docs")` returns the names of all documents from the database named `docs`.

db:list-details

Signatures `db:list-details()` as element(database)*
`db:list-details($db as item())` as element(resource)*
`db:list-details($db as item(), $path as xs:string)` as element(resource)*

Summary Returns an `element` sequence with the names of all databases together with their database path, the number of stored resources and the date of modification.

If `$db` is specified, all documents and raw files of the specified database together with their content-type, the modification date and the resource type are returned.

The list of resources can be further restricted by the `$path` argument.

Examples

- `db:list-details("docs")` returns the names plus additional data of all documents from the database named `docs`.

db:open

Signatures `db:open($db as item())` as document-node()*
`db:open($db as item(), $path as xs:string)` as document-node()*

Summary Returns a sequence with all document nodes contained in the database specified by `$db`.
The document nodes to be returned can be restricted by the `$path` argument.

Examples

- `db:open("docs")` returns all documents from the database named `docs`.
- `db:open("docs", "one")` returns all documents from the database named `docs` in the subpath `one`.

db:open-id

Signatures `db:open-id($db as item(), $id as xs:integer)` as node()

Summary Opens the database specified by `$db` and returns the node with the specified `$id` value.

Each database node has a persistent `id`, which remains valid after update operations. If no updates are performed, the `pre` value can be requested, which provides access to database nodes in constant time.

db:open-pre

Signatures `db:open-pre($db as item(), $pre as xs:integer)` as node()

Summary Opens the database specified by `$db` and returns the node with the specified `$pre` value.

The `pre` value provides access to a database node in constant time, but it is *transient*, i.e., it may change when database updates are performed.

Errors **BASX0004** is raised if the specified `$pre` value does not exist in the database.

Examples

- `db:open-pre("docs", 0)` returns the first database node from the database named `docs`.

db:system

Signatures `db:system()` as element(system)

Summary Returns information on the database system, such as the database path and current database settings.

Read Operations

db:attribute

Signatures `db:attribute($db as item(), $string as item())` as attribute()*

`db:attribute($db as item(), $string as item(), $attname as xs:string)` as attribute()*

Summary Returns all attribute nodes that have `$string` as string value. If available, the value index is used to speed up evaluation. If `$attname` is specified, the resulting attribute nodes are filtered by their attribute name.

Examples

- `db:attribute("DB", "QUERY", "id")/..` returns the parents of all `id` attribute nodes of the database `DB` that have `QUERY` as string value.

db:attribute-range

Introduced with Version 7.2.1:

Signatures `db:attribute-range($db as item(), $min as xs:string, $max as xs:string)` as text()*

`db:attribute-range($db as item(), $min as xs:string, $max as xs:string, $attname as xs:string)` as attribute()*

Summary Returns all attributes the string values of which are larger than or equal to `$min` and smaller than or equal to `$max`. If available, the value index is used to speed up evaluation.

Examples

- `db:attribute-range("DB", "id456", "id473", 'id')` returns all `@id` attributes of the database `DB` that have a string value in between `id456` and `id473`.

db:fulltext

Signatures `db:fulltext($db as item(), $text as item())` as text()*

Summary Returns all text nodes from the full-text index that contain the string `$text`.

The index full-text options will be applied here: if the index terms have been stemmed, the search string will be stemmed as well.

Errors `BASX0001` is raised if the index is not available.

Examples

- `db:fulltext("DB", "QUERY")` returns all text nodes of the database `DB` that contain the string `QUERY`.

db:node-id

Signatures `db:node-id($nodes as node()*) as xs:integer*`

Summary Returns the *id* values of all database nodes specified by *\$nodes*.

Each database node has a persistent *id*, which remains valid after update operations. If no updates are performed, the *pre* value can be requested, which provides access to database nodes in constant time.

db:node-pre

Signatures `db:node-pre($nodes as node()*) as xs:integer*`

Summary Returns the *pre* values of all database nodes specified by *\$nodes*.

The *pre* value provides access to a database node in constant time, but it is *transient*, i.e., it may change when database updates are performed.

Examples

- `db:node-pre(doc("input"))` returns 0 if the database `input` contains a single document.

db:retrieve

Signatures `db:retrieve($db as item(), $path as xs:string) as xs:base64Binary`

Summary Returns a binary database resource addressed by *\$db* and *\$path*.

Errors **FODC0002** is raised if the addressed resource is not found or cannot be retrieved.

Examples

- `declare option output:method 'raw';`
`db:retrieve("DB", "music/01.mp3")` returns the specified audio file as raw data.

db:text

Signatures `db:text($db as item(), $string as item()) as text()*`

Summary Returns all text nodes that have *\$string* as their string value. If available, the value index is used to speed up evaluation.

Examples

- `db:text("DB", "QUERY")/..` returns the parents of all text nodes of the database `DB` that match the string `QUERY`.

db:text-range

Introduced with Version 7.2.1:

Signatures `db:text-range($db as item(), $min as xs:string, $max as xs:string) as text()*`

Summary Returns all text nodes that are located in between the *\$min* and *\$max* strings. If available, the value index is used to speed up evaluation.

Examples

- `db:text-range("DB", "2000", "2001")` returns all text nodes of the database `DB` that are found in between 2000 and 2001.

Updates

All functions in this section are treated as **updating**: they are not immediately executed, but queued on a *pending update list*, which is processed after the actual query has been evaluated. The XQuery Update page gives more insight into the relevant concepts.

db:add

Signatures `db:add($db as item(), $input as item()) as empty-sequence()`
`db:add($db as item(), $input as item(), $path as xs:string) as empty-sequence()`

Summary Adds documents specified by `$input` to the database `$db` and the specified `$path`.

Errors **FODC0002** is raised if `$input` is a string representing a path, which cannot be read.
FOUP0001 is raised if `$input` is not a string and not a document node.

Examples

- `db:add("DB", "/home/dir/doc.xml")` adds the file `/home/dir/doc.xml` to the database `DB`.
- `db:add("DB", "<a/>", "doc.xml")` adds a document with content `<a/>` to the database `DB` under the name `doc.xml`.
- `db:add("DB", document { <a/> }, "doc.xml")` adds the document node to the database `DB` under the name `doc.xml`.
- `db:add("DB", "/home/dir", "docs/dir")` adds all documents in `/home/dir` to the database `DB` under the path `docs/dir`.

db:delete

Signatures `db:delete($db as item(), $path as xs:string) as empty-sequence()`

Summary Deletes document(s), specified by `$path`, from the database `$db`.

Examples

- `db:delete("DB", "docs/dir/doc.xml")` deletes the document `docs/dir/doc.xml` in the database `DB`.
- `db:delete("DB", "docs/dir")` deletes all documents with paths beginning with `docs/dir` in the database `DB`.

db:optimize

Signatures `db:optimize($db as item()) as empty-sequence()`
`db:optimize($db as item(), $all as xs:boolean) as empty-sequence()`

Summary Optimizes the meta data and indexes of the database `$db`.
 If `$all` is set to `true()`, the complete database will be rebuilt.

Errors **BASX0014** is raised if an error occurs during optimizing the data structures.
BASX0015 is raised if the `$all` flag is set to `true()`, but the database is an in-memory database.
BASX0016 is raised if the database `$db` is in use by other user(s).

Examples

- `db:optimize("DB")` optimizes the database structures of the database `DB`.
- `db:optimize("DB", true())` optimizes all database structures of the database `DB`.

db:rename

Signatures `db:rename($db as item(), $path as xs:string, $newpath as xs:string) as empty-sequence()`

Summary Renames document(s), specified by `$path` to `$newpath` in the database `$db`.

Errors **BASX0013** is raised if new document name(s) will be empty.

Examples

- `db:rename("DB", "docs/dir/doc.xml", "docs/dir/newdoc.xml")` renames the document `docs/dir/doc.xml` to `docs/dir/newdoc.xml` in the database `DB`.
- `db:rename("DB", "docs/dir", "docs/newdir")` renames all documents with paths beginning with `docs/dir` to paths beginning with `docs/newdir` in the database `DB`.

db:replace

Signatures `db:replace($db as item(), $path as xs:string, $input as item()) as empty-sequence()`

Summary Replaces a document, specified by `$path`, in the database `$db` with the content of `$input`.

Errors **BASX0012** is raised if `$path` is not a single document path.
FODC0002 is raised if `$input` is a string representing a path, which cannot be read.
FOUP0001 is raised if `$input` is not a string and not a document node.

Examples

- `db:replace("DB", "docs/dir/doc.xml", "/home/dir/doc.xml")` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with the content of the file `/home/dir/doc.xml`.
- `db:replace("DB", "docs/dir/doc.xml", "<a/>")` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with `<a/>`.
- `db:replace("DB", "docs/dir/doc.xml", document { <a/> })` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with the specified document node.

db:store

Signatures `db:store($db as item(), $path as xs:string, $data as item()) as empty-sequence()`

Summary Stores a binary resource specified by `$data` at the location specified by `$path`.

Errors **FOUP0002** is raised if the resource cannot be stored at the specified location.

Examples

- `db:store("DB", "video/sample.mov", file:read-binary('video.mov'))` stores the addressed video file at the specified location.

db:output

Introduced with Version 7.2.1:

Signatures `db:output($data as item(*) as empty-sequence()`

Summary This function can be used to both perform updates and return results in a single query. The argument of the function will be evaluated, and the resulting value will be cached and returned after the updates on the *pending update list* have been processed. The function can only be used together with updating expressions; if the function is called within a transform expression, its results will be discarded.

Examples

- `db:output("Prices have been deleted."), delete node //price` deletes all price elements in a database and returns an info message.

Helper Functions

db:exists

Signatures `db:exists($db as item()) as xs:boolean`
`db:exists($db as item(), $path as xs:string) as xs:boolean`

Summary Checks if the specified database or resource exists. `false` is returned if a database directory is specified.

Examples

- `db:exists("DB")` returns `true` if the database `DB` exists.
- `db:exists("DB", "resource")` returns `true` if `resource` is an XML document or a raw file.

db:is-raw

Signatures `db:is-raw($db as item(), $path as xs:string) as xs:boolean`

Summary Checks if the specified resource exists and if it is a raw file.

Examples

- `db:is-raw("DB", "music/01.mp3")` returns `true`.

db:is-xml

Signatures `db:is-xml($db as item(), $path as xs:string) as xs:boolean`

Summary Checks if the specified resource exists and if it is an XML document.

Examples

- `db:is-xml("DB", "dir/doc.xml")` returns `true`.

db:content-type

Signatures `db:content-type($db as item(), $path as xs:string) as xs:string`

Summary Retrieves the content type of the resource specified by `$path`.

The file extension is used to recognize the content-type of a resource stored in the database. Content-type `application/xml` will be returned for any XML document stored in the database, regardless of its file name extension.

Errors **FODC0002** is raised if the addressed resource is not found or cannot be retrieved.

Examples

- `db:content-type("DB", "docs/doc01.pdf")` returns `application/pdf`.
- `db:content-type("DB", "docs/doc01.xml")` returns `application/xml`.
- `db:content-type("DB", "docs/doc01")` returns `application/xml`, if `db:is-xml("DB", "docs/doc01")` returns `true`.

db:event

Signatures `db:event($name as xs:string, $query as item()) as empty-sequence()`

Summary Executes a `$query` and sends the resulting value to all clients watching the Event with the specified `$name`. The query may also perform updates; no event will be sent to the client that fired the event.

Errors **BASX0009** is raised if the specified event is unknown.
SEPM0016 is raised if serialization errors occurred while sending the value.

Changelog

Version 7.2.1

- Added: `db:text-range`, `db:attribute-range`, `db:output`

Version 7.1

- Added: `db:list-details`, `db:content-type`
- Updated: `db:info`, `db:system`, `db:retrieve`

Version 7.0

- Added: `db:retrieve`, `db:store`, `db:exists`, `db:is-raw`, `db:is-xml`
 - Updated: `db:list`, `db:open`, `db:add`
-

File Module

This XQuery Module contains functions and variables related to file system operations, such as listing, reading, or writing files. All functions are preceded by the `file:` prefix, which is linked to the statically declared `http://expath.org/ns/file` namespace. This module has been aligned with the latest EXPath File Module ^[2] draft from 2011 (expected to be online soon).

`$file:directory-separator`

Signatures `$file:directory-separator` as `xs:string`

Summary This variable returns the directory separator used by the operating system, such as "/" or "\".

`$file:path-separator`

Signatures `$file:path-separator` as `xs:string`

Summary This variable returns the path separator used by the operating system, such as ";" or ":".

`file:exists`

Signatures `file:exists`(\$path as `xs:string`) as `xs:boolean`

Summary Returns an `xs:boolean` indicating whether a file or directory specified by \$path exists in the file system.

`file:is-directory`

Signatures `file:is-directory`(\$path as `xs:string`) as `xs:boolean`

Summary Returns an `xs:boolean` indicating whether the argument \$path points to an existing directory.

`file:is-file`

Signatures `file:is-file`(\$path as `xs:string`) as `xs:boolean`

Summary Returns an `xs:boolean` indicating whether the argument \$path points to an existing file.

file:last-modified

Signatures `file:last-modified($path as xs:string) as xs:dateTime`

Summary Retrieves the timestamp of the last modification of the file or directory specified by `$path`.

Errors **FOFL0001** is raised if the specified path does not exist.

file:size

Signatures `file:size($file as xs:string) as xs:integer`

Summary Returns the size, in bytes, of the file specified by `$path`.

Errors **FOFL0001** is raised if the specified file does not exist.
FOFL0004 is raised if the specified file points to a directory.

file:base-name

Signatures `file:base-name($path as xs:string) as xs:string`

`file:base-name($path as xs:string, $suffix as xs:string) as xs:string`

Summary Returns the base-name of the path specified by `$path`, which is the component after the last directory separator.

If `$suffix` is specified, it will be trimmed from the end of the result.

file:dir-name

Signatures `file:dir-name($path as xs:string) as xs:string`

Summary Returns the parent directory of the path specified by `$path`, which is the component before the last directory separator.

file:path-to-native

Signatures `file:path-to-native($path as xs:string) as xs:string`

Summary Transforms the `$path` argument to its native representation on the operating system.

Errors **FOFL0000** is raised if the specified path cannot be transformed to its native representation.

file:resolve-path

Signatures `file:resolve-path($path as xs:string) as xs:string`

Summary Transforms the `$path` argument to an absolute operating system path.

file:path-to-uri

Signatures `file:path-to-uri($path as xs:string) as xs:string`

Summary Transforms the path specified by `$path` into a URI with the `file://` scheme.

file:list

Signatures `file:list` (\$directory as xs:string) as xs:string*
`file:list` (\$directory as xs:string, \$recursive as xs:boolean) as xs:string*
`file:list` (\$directory as xs:string, \$recursive as xs:boolean, \$pattern as xs:string) as xs:string*

Summary Lists all files and directories found in the specified \$directory. The returned paths are relative to the provided path. The optional parameter \$recursive specifies whether sub-directories will be traversed, too. The optional parameter \$pattern defines a file name pattern in the glob syntax^[1]. If present, only those files and directories are returned that correspond to the pattern. Several patterns can be separated with a comma (,).

Errors **FOFL0003** is raised if the specified path does not point to a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:create-directory

Signatures `file:create-directory` (\$directory as xs:string) as empty-sequence()

Summary Recursively creates the directories specified by \$directory.

Errors **FOFL0002** is raised if a file with the same path already exists.
FOFL0000 is raised if the operation fails for some other reason.

file:delete

Updated with Version 7.2.1: \$recursive parameter added to prevent sub-directories from being accidentally deleted.

Signatures `file:delete` (\$path as xs:string) as empty-sequence()
`file:delete` (\$path as xs:string, \$recursive as xs:boolean) as empty-sequence()

Summary Recursively deletes a file or directory specified by \$path. The optional parameter \$recursive specifies whether sub-directories will be deleted, too.

Errors **FOFL0001** is raised if the specified path does not exist.
FOFL0000 is raised if the operation fails for some other reason.

file:read-text

Signatures `file:read-text` (\$path as xs:string) as xs:string
`file:read-text` (\$path as xs:string, \$encoding as xs:string) as xs:string

Summary Reads the textual contents of the file specified by \$path and returns it as a xs:string. The optional parameter \$encoding defines the encoding of the file.

Errors **FOFL0001** is raised if the specified file does not exist.
FOFL0004 is raised if the specified path is a directory.
FOFL0005 is raised if the specified encoding is not supported, or unknown.
FOFL0000 is raised if the operation fails for some other reason.

file:read-text-lines

Signatures `file:read-text-lines`(\$path as xs:string) as xs:string
`file:read-text-lines`(\$path as xs:string, \$encoding as xs:string) as xs:string*

Summary Reads the textual contents of the file specified by \$path and returns it as a sequence of xs:string items. The optional parameter \$encoding defines the encoding of the file.

Errors **FOFL0001** is raised if the specified file does not exist.
FOFL0004 is raised if the specified path is a directory.
FOFL0005 is raised if the specified encoding is not supported, or unknown.
FOFL0000 is raised if the operation fails for some other reason.

file:read-binary

Signatures `file:read-binary`(\$path as xs:string) as xs:base64Binary

Summary Reads the binary content of the file specified by \$path and returns as a xs:base64Binary.

Errors **FOFL0001** is raised if the specified file does not exist.
FOFL0004 is raised if the specified path is a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:write

Signatures `file:write`(\$path as xs:string, \$items as item()*) as empty-sequence()
`file:write`(\$path as xs:string, \$items as item()*, \$params as xs:node()*) as empty-sequence()

Summary Writes a sequence of \$items to a file specified by \$path. If the specified file already exists, it will be overwritten. The optional argument \$params is used to set the serialization parameters (see [Serialization](#) for more details). It can be specified as

- `element(serialization-parameters):<serialization-parameters/>` must be used as root element, and the parameters are specified as child nodes, with the element name representing the serialization parameter and the attribute value representing its value:

```
<serialization-parameters xmlns="[2]">
  <method value='xml' />
  <cdata-section-elements value="div" />
  ...
</serialization-parameters>
```

- `map` structure: all parameters can be directly represented as key/value pairs:
`map { "method" := "xml", "cdata-section-elements" := "div", ... }`

Errors **FOFL0004** is raised if the specified path is a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:write-binary

Signatures `file:write-binary($path as xs:string, $items as xs:base64Binary*) as empty-sequence()`

Summary Writes a sequence of `xs:base64Binary` `$items` to a file specified by `$path`. If the specified file already exists, it will be overwritten.

Errors **FOFL0004** is raised if the specified path is a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:append

Signatures `file:append($path as xs:string, $items as item()*) as empty-sequence()`
`file:append($path as xs:string, $items as item()*, $params as xs:node()*) as empty-sequence()`

Summary Appends a sequence of `$items` to a file specified by `$path`. If the specified file does not exist, a new file is created.

Errors **FOFL0004** is raised if the specified path is a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:append-binary

Signatures `file:append-binary($path as xs:string, $items as xs:base64Binary*) as empty-sequence()`

Summary Appends a sequence of `xs:base64Binary` `$items` to a file specified by `$path`. If the specified file does not exist, a new file is created.

Errors **FOFL0004** is raised if the specified path is a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:copy

Signatures `file:copy($source as xs:string, $target as xs:string) as empty-sequence()`

Summary Copies a file specified by `$source` to the file or directory specified by `$target`. If the target represents an existing file, it will be overwritten. No operation will be performed if the source and target path are equal.

Errors **FOFL0001** is raised if the specified source does not exist.
FOFL0002 is raised if the specified source is a directory and the target is a file.
FOFL0003 is raised if the parent of the specified target is not a directory.
FOFL0000 is raised if the operation fails for some other reason.

file:move

Signatures `file:move($source as xs:string, $target as xs:string) as empty-sequence()`

Summary Moves or renames the file or directory specified by `$source` to the path specified by `$target`. No operation will be performed if the source and target path are equal.

Errors

- FOFL0001** is raised if the specified source does not exist.
- FOFL0002** is raised if the specified source is a directory and the target is a file.
- FOFL0003** is raised if the parent of the specified target is no directory.
- FOFL0000** is raised if the operation fails for some other reason.

Changelog

Version 7.2.1

- Updated: `file:delete: $recursive` parameter added to prevent sub-directories from being accidentally deleted.

References

- [1] [http://en.wikipedia.org/wiki/Glob_\(programming\)](http://en.wikipedia.org/wiki/Glob_(programming))
- [2] <http://www.w3.org/2010/xslt-xquery-serialization>

Full-Text Module

This XQuery Module extends the W3C Full Text Recommendation ^[1] with some useful functions: The index can be directly accessed, full-text results can be marked with additional elements, or the relevant parts can be extracted. Moreover, the score value, which is generated by the `contains text` expression, can be explicitly requested from items. All functions are introduced with the `ft:` prefix, which is linked to the statically declared `http://basex.org/modules/ft` namespace.

Functions

`ft:search`

Signatures `ft:search($node as node(), $text as xs:string) as text()`

Summary Performs a full-text index request on the specified XML node and returns all text nodes that contain the string `$text`. The index full-text options are used for searching, i.e., if the index terms were stemmed, the search string will be stemmed as well.

Errors

- BASX0001** is raised if the full-text index is not available.
- BASX0002** is raised if a referenced node is not stored in a database (i.e., references a main-memory XML fragment).

Examples

- `ft:search(., "QUERY")` returns all text nodes of the currently opened database that contain the string "QUERY".

`ft:mark`

Signatures `ft:mark`(\$nodes as node()*) as node()*
`ft:mark`(\$nodes as node()*, \$tag as xs:string) as node()*

Summary Puts a marker element around the resulting \$nodes of a full-text index request. The default tag name of the marker element is `mark`. An alternative tag name can be chosen via the optional \$tag argument. Note that the XML node to be transformed must be an internal "database" node. The `transform` expression can be used to apply the method to a main-memory fragment (see example).

Errors **BASX0002** is raised if a referenced node is not stored in a database (i.e., references a main-memory XML fragment). **FOCA0002** is raised if \$name is no valid QName.

Examples • The following query returns `<XML><mark>hello</mark> world</XML>`, if one text node of the database `DB` has the value "hello world":

```
ft:mark(db:open('DB')//*[text() contains text 'hello'])
```

• The following expression returns `<p>word</p>`:

```
copy $p := <p>word</p>
modify ()
return ft:mark($p[text() contains text 'word'], 'b')
```

ft:extract

Signatures `ft:extract`(\$nodes as node()*) as node()*
`ft:extract`(\$nodes as node()*, \$tag as xs:string) as node()*
`ft:extract`(\$nodes as node()*, \$tag as xs:string, \$length as xs:integer) as node()*

Summary Extracts and returns relevant parts of full-text results. It puts a marker element around the resulting \$nodes of a full-text index request and chops irrelevant sections of the result. The default tag name of the marker element is `mark`. An alternative tag name can be chosen via the optional \$tag argument. The default length of the returned text is 150 characters. An alternative length can be specified via the optional \$length argument. Note that the effective text length may differ from the specified text due to formatting and readability issues.

Errors **BASX0002** is raised if a referenced node is not stored in a database (i.e., references a main-memory XML fragment). **FOCA0002** is raised if \$name is no valid QName.

Examples • The following query may return `<XML>...hello...<XML>` if a text node of the database `DB` contains the string "hello world":

```
ft:extract(db:open('DB')//*[text() contains text 'hello'], 'b', 1)
```

ft:count

Signatures `ft:count`(\$nodes as node()*) as xs:integer

Summary Returns the number of occurrences of the search terms specified in a full-text expression.

Errors **BASX0002** is raised if a referenced node is not stored in a database (i.e., references a main-memory XML fragment).

Examples • `ft:count(//*[text() contains text 'QUERY'])` returns the `xs:integer` value 2 if a document contains two occurrences of the string "QUERY".

ft:score

Signatures `ft:score($item as item(*) as xs:double*`

Summary Returns the score values (0.0 - 1.0) that have been attached to the specified items. 0 is returned a value if no score was attached.

Examples • `ft:score('a' contains text 'a')` returns the `xs:double` value 1.

ft:tokens

Signatures `ft:tokens($db as item()) as element(value)*`

`ft:tokens($db as item(), $prefix as xs:string) as element(value)*`

Summary Returns all full-text tokens stored in the index, along with their numbers of occurrences. `$db` may either be an `xs:string`, denoting the database name, or a node stored in the database.

If `$prefix` is specified, the returned nodes will be refined to the strings starting with that prefix. The prefix will be tokenized according to the full-text used for creating the index.

Errors **BASX0001** is raised if the full-text index is not available.

BASX0002 is raised if `$db` references a node that is not stored in a database (i.e., references a main-memory XML fragment).

BASX0003 is raised if the addressed database cannot be opened.

ft:tokenize

Signatures `ft:tokenize($input as xs:string) as xs:string*`

Summary Tokenizes the given `$input` string, using the current default full-text options.

Examples • `ft:tokenize("No Doubt")` returns the two strings `no` and `doubt`.

• `declare ft-option using stemming; ft:tokenize("GIFTS")` returns a single string `gift`.

Changelog

Version 7.1

- Added: `ft:tokens`, `ft:tokenize`

References

[1] <http://www.w3.org/TR/xpath-full-text-10>

HTTP Module

This XQuery Module contains a single function to send HTTP requests and handle HTTP responses. The function `send-request`, which is introduced with the `http:` prefix, is linked to the statically declared `http://expath.org/ns/http-client` namespace and based on the EXPath HTTP Client Module ^[3]:

`http:send-request`

Signatures `http:send-request` (`$request` as element(`http:request`)?, `$href` as xs:string?, `$bodies` as item(*) as item()+
`http:send-request` (`$request` as element(`http:request`)) as item()+
`http:send-request` (`$request` as element(`http:request`)?, `$href` as xs:string?) as item()+

Summary Sends an HTTP request and interprets the corresponding response. `$request` contains the parameters of the HTTP request such as HTTP method and headers. In addition to this it can also contain the URI to which the request will be sent and the body of the HTTP method. If the URI is not given with the parameter `$href`, its value in `$request` is used instead. The structure of `http:request` element follows the EXPath ^[3] specification.

Notes The attribute `auth-method` of `$request` is not considered in our implementation because we are handling only basic authentication.

Examples

Status Only

Simple GET request. As the attribute `status-only` is set to `true`, only the response element is returned.

Query:

```
http:send-request(<http:request method='get' status-only='true'/>, 'http://basex.org')
```

Result:

```
<http:response status="200" message="OK">
  <http:header name="Date" value="Mon, 14 Mar 2011 20:55:53 GMT"/>
  <http:header name="Content-Length" value="12671"/>
  <http:header name="Expires" value="Mon, 14 Mar 2011 20:57:23 GMT"/>
  <http:header name="Set-Cookie" value="fe_typo_user=d10c9552f9a784d1a73f8b6ebdf5ce63; path=/" />
  <http:header name="Connection" value="close"/>
  <http:header name="Content-Type" value="text/html; charset=utf-8"/>
  <http:header name="Server" value="Apache/2.2.16"/>
  <http:header name="X-Powered-By" value="PHP/5.3.5"/>
  <http:header name="Cache-Control" value="max-age=90"/>
  <http:body media-type="text/html; charset=utf-8"/>
</http:response>
```

Google Homepage

Retrieve Google search home page. TagSoup ^[2] must be referenced in the class path in order to parse html.

Query:

```
http:send-request (<http:request method='get' href='http://www.google.com' />)
```

Result:

```
<http:response status="200" message="OK">
  <http:header name="Date" value="Mon, 14 Mar 2011 22:03:25 GMT"/>
  <http:header name="Transfer-Encoding" value="chunked"/>
  <http:header name="Expires" value="-1"/>
  <http:header name="X-XSS-Protection" value="1; mode=block"/>
  <http:header name="Set-Cookie" value="...; expires=Tue, 13-Sep-2011 22:03:25 GMT; path=/; domain=.google.ch; HttpOnly"/>
  <http:header name="Content-Type" value="text/html; charset=ISO-8859-1"/>
  <http:header name="Server" value="gws"/>
  <http:header name="Cache-Control" value="private, max-age=0"/>
  <http:body media-type="text/html; charset=ISO-8859-1"/>
</http:response>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"/>
    <title>Google</title>
    <script>window.google={kEI:"rZB-
    ...
  </script>
  </center>
</body>
</html>
```

SVG Data

Content-type ending with +xml, e.g. image/svg+xml.

Query:

```
http:send-request (<http:request method='get' />, 'http://upload.wikimedia.org/wikipedia/commons/6/6b/Bitmap_VS_SVG.svg')
```

Result:

```
<http:response status="200" message="OK">
  <http:header name="ETag" value="W/"11b6d-4ba15ed4"" />
  <http:header name="Age" value="9260"/>
  <http:header name="Date" value="Mon, 14 Mar 2011 19:17:10 GMT"/>
  <http:header name="Content-Length" value="72557"/>
  <http:header name="Last-Modified" value="Wed, 17 Mar 2010 22:59:32 GMT"/>
  <http:header name="Content-Type" value="image/svg+xml"/>
  <http:header name="X-Cache-Lookup" value="MISS from knsq22.knams.wikimedia.org:80"/>
  <http:header name="Connection" value="keep-alive"/>
  <http:header name="Server" value="Sun-Java-System-Web-Server/7.0"/>
  <http:header name="X-Cache" value="MISS from knsq22.knams.wikimedia.org"/>
```

```

<http:body media-type="image/svg+xml"/>
</http:response>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" width="1063" height="638">
  <defs>
    <linearGradient id="lg0">
      <stop stop-color="#3333ff" offset="0"/>
      <stop stop-color="#3f3fff" stop-opacity="0" offset="1"/>
    </linearGradient>
    ...
  </svg>

```

POST Request

POST request to the BaseX REST Service, specifying a username and password.

Query:

```

let $request :=
  <http:request href='http://localhost:8984/rest'
    method='post' username='admin' password='admin' send-authorization='true'>
    <http:body media-type='application/xml'>
      <query xmlns="http://basex.org/rest">
        <text><![CDATA[
          <html{
            for $i in 1 to 3
              return <div>Section { $i }</div>
          }</html>
        ]]></text>
      </query>
    </http:body>
  </http:request>
return http:send-request ($request)

```

Result:

```

<http:response xmlns:http="http://expath.org/ns/http-client" status="200" message="OK">
  <http:header name="Content-Length" value="135"/>
  <http:header name="Content-Type" value="application/xml"/>
  <http:header name="Server" value="Jetty (6.1.26)"/>
  <http:body media-type="application/xml"/>
</http:response>
<html>
  <div>Section 1</div>
  <div>Section 2</div>
  <div>Section 3</div>
</html>

```

Higher-Order Functions Module

This XQuery Module adds some useful higher-order functions, additional to the Higher-Order Functions provided by the official specification. All functions are introduced with the `hof:` prefix, which is linked to the statically declared `http://basex.org/modules/hof` namespace.

Functions

hof:id

Signatures `hof:id($expr as item()*) as item()*`

Summary Returns its argument unchanged. This function isn't useful on its own, but can be used as argument to other higher-order functions.

Examples 5 4 3 2 1

hof:const

Signatures `hof:const($expr as item()*, $ignored as item()*) as item()*`

Summary Returns its first argument unchanged and ignores the second. This function isn't useful on its own, but can be used as argument to other higher-order functions, e.g. when a function combining two values is expected and one only wants to retain the left one.

Examples

- `hof:const(42, 1337)` returns 42.
- With higher-order functions:

```
let $zip-sum := function($f, $seq1, $seq2) {
  sum(map-pairs($f, $seq1, $seq2))
}
let $sum-all := $zip-sum(function($a, $b) { $a + $b }, ?, ?),
    $sum-left := $zip-sum(hof:const#2, ?, ?)
return (
  $sum-all((1, 1, 1, 1, 1), 1 to 5),
  $sum-left((1, 1, 1, 1, 1), 1 to 5)
)
```

- Another use-case: When inserting a key into a map, `$f` decides how to combine the new value with a possibly existing old one. `hof:const` here means ignoring the old value, so that's normal insertion.

```
let $insert-with := function($f, $map, $k, $v) {
  let $old := $map($k),
      $new := if($old) then $f($v, $old) else $v
  return map:new(($map, map{ $k := $new }))
}
let $map := map{ 'foo' := 1 }
let $add := $insert-with(function($a, $b) { $a + $b }, ?, ?, ?),
    $insert := $insert-with(hof:const#2, ?, ?, ?)
return (
  $add($map, 'foo', 2)('foo'),
  $insert($map, 'foo', 42)('foo')
)
```

returns 3 42

hof:fold-left1

Signatures `hof:fold-left1`(\$f as function(item()* , item()) as item()* , \$seq as item()+) as item()*

Summary Works the same as `fn:fold-left`(\$f, \$seed, \$seq), but doesn't need a seed, because the sequence must be non-empty.

Errors `XPTY0004` if \$seq is empty

- Examples**
- `hof:fold-left1(function($a, $b) { $a + $b }, 1 to 10)` returns 55.
 - `hof:fold-left1(function($a, $b) { $a + $b }, ())` throws `XPTY0004`, because \$seq has to be non-empty.

hof:until

Signatures `hof:until`(\$pred as function(item()*) as xs:boolean, \$f as function(item()*) as item()* , \$start as item()*) as item()*

Summary Applies the function \$f to the initial value \$start until the predicate \$pred applied to the result returns true().

- Examples**
- `hof:until(function($x) { $x ge 1000 }, function($y) { 2 * $y }, 1)` returns 1024.
 - Calculating the square-root of a number by iteratively improving an initial guess:

```
let $sqrt := function($x as xs:double) as xs:double {
  hof:until(
    function($res) { abs($res * $res - $x) < 0.00001 },
    function($guess) { ($guess + $x div $guess) div 2 },
    $x
  )
}
return $sqrt(25)
```

returns 5.000000000053722.

hof:top-k-by

Introduced with [Version 7.2](#):

Signatures `hof:top-k-by`(\$seq as item()* , \$sort-key as function(item()) as item() , \$k as xs:integer) as item()*

Summary Returns the \$k items in \$seq that are greatest when sorted by the result of \$f applied to the item. The function is a much more efficient implementation of the following scheme:

```
(
  for $x in $seq
  order by $sort-key($x) descending
  return $x
)[position() <= $k]
```

Errors `XPTY0004` if \$sort-key doesn't return exactly one item

- Examples**
- `hof:top-k-by(1 to 1000, hof:id#1, 5)` returns 1000 999 998 997 996
 - `hof:top-k-by(1 to 1000, function($x) { -$x }, 3)` returns 1 2 3
 - `hof:top-k-by(<x a='1' b='2' c='3'/>/@*, xs:integer#1, 2)/node-name()` returns c b

hof:top-k-with

Introduced with Version 7.2:

Signatures `hof:top-k-with($seq as item()*, $lt as function(item(), item()) as xs:boolean, $k as xs:integer) as item()*`

Summary Returns the `$k` items in `$seq` that are greatest when sorted in the order of the *less-than* predicate `$lt`. The function is a general version of `hof:top-k-by($seq, $sort-key, $k)`.

Examples

- `hof:top-k-with(1 to 1000, function($a, $b) { $a lt $b }, 5)` returns `1000 999 998 997 996`
- `hof:top-k-with(-5 to 5, function($a, $b) { abs($a) gt abs($b) }, 5)` returns `0 1 -1 2 -2`

Changelog

Version 7.2

- Added: `hof:top-k-by`, `hof:top-k-with`
- Removed: `hof:iterate`

Version 7.0

- module added

Index Module

This XQuery Module provides functions for displaying information stored in the database index structures. All functions are introduced with the `index:` prefix, which is linked to the statically declared <http://basex.org/modules/index> namespace.

Commonalities

All functions share `$db` as argument, which may either reference a string, denoting the name of the addressed database, or a node from an already opened database. The following errors may be raised by all functions:

- **BASX0001** is raised if the index required by a function is not available.
- **BASX0002** is raised if `$db` references a node that is not stored in a database (i.e., references a main-memory XML fragment).
- **BASX0003** is raised if the addressed database cannot be opened.

Functions

index:facets

Signatures `index:facets($db as item()) as xs:string`
`index:facets($db as item(), $type as xs:string) as xs:string`

Summary Returns information about possible facets and facet values on a database in document structure format.
If `$type` is specified as `flat`, the function returns this information in a flat summarized version.

Examples

- `index:facets("DB")` returns information about facets and facet values on the database `DB` in document structure.
- `index:facets("DB", "flat")` returns information about facets and facet values on the database `DB` in a summarized flat structure.

index:texts

Signatures `index:texts($db as item()) as element(value)*`
`index:texts($db as item(), $prefix as xs:string) as element(value)*`

Summary Returns all strings stored in the text index, along with their number of occurrences.
If `$prefix` is specified, the returned nodes will be refined to the strings starting with that prefix.

index:attributes

Signatures `index:attributes($db as item()) as element(value)*`
`index:attributes($db as item(), $prefix as xs:string) as element(value)*`

Summary Returns all strings stored in the attribute index, along with their number of occurrences.
If `$prefix` is specified, the returned nodes will be refined to the strings starting with that prefix.

index:element-names

Signatures `index:element-names($db as item()) as element(value)*`

Summary Returns all element names stored in the index, along with their number of occurrences.

index:attribute-names

Signatures `index:attribute-names($db as item()) as element(value)*`

Summary Returns all attribute names stored in the index, along with their number of occurrences.

Changelog

The module was introduced with [Version 7.1](#).

JSON Module

This XQuery Module contains functions to parse and serialize JSON documents. All functions are preceded by the `json:` prefix, which is linked to the statically declared `http://basex.org/modules/json` namespace.

JSON (JavaScript Object Notation) ^[1] is a popular data exchange format for applications written in JavaScript. As there are notable differences between JSON and XML, no mapping exists that guarantees a lossless, bidirectional conversion between JSON and XML. For this reason, we offer two sets of functions in this module:

JSON Functions

`json:parse` and `json:serialize` facilitate a lossless conversion from JSON to XML and back. The transformation is based on the following rules:

1. The resulting document has a `<json/>` root node.
2. Names (keys) of objects are represented as elements:
 1. Empty names are represented by a single underscore (`<_> . . . </_>`).
 2. Underscore characters are rewritten to two underscores (`__`).
 3. A character that cannot be represented as NCName character is rewritten to an underscore and its four-digit Unicode.
3. As the members of arrays have no names, `<value/>` is used as element name.
4. JSON values are represented as text nodes.
5. The types of values are represented in attributes:
 1. The value types *number*, *boolean*, *null*, *object* and *array* are represented by a `type` attribute.
 2. The *string* type is omitted, as it is treated as default type.
 3. If a name has the same type throughout the document, the `type` attribute will be omitted. Instead, the name will be listed in additional, type-specific attributes in the root node. The attributes are named by their type in the plural (*numbers*, *booleans*, *nulls*, *objects* and *arrays*), and the attribute value contains all names with that type, separated by whitespaces.

`json:parse`

Signatures `json:parse($input as xs:string()) as element(json)`

Summary Converts the JSON document specified by `$input` to XML, and returns the result as `element(json)` instance. The converted XML document is both well readable and lossless, i.e., the converted document can be serialized back to the original JSON representation.

Errors **BASX0015** is raised if the specified input cannot be parsed as JSON document.

`json:serialize`

Signatures `json:serialize($input as node()) as xs:string()`

Summary Serializes the node specified by `$input` as JSON, and returns the result as `xs:string` instance. The serialized node must conform to the syntax specified by the `json:parse()` function.

XML documents can also be serialized as JSON if the Serialization Option "method" is set to "json".

Errors **BASX0016** is raised if the specified node cannot be serialized as JSON document.

Examples

Example 1: Adds all JSON documents in a directory to a database

Query:

```
let $database := "database"
for $name in file:list('.', false(), '*.json')
let $file := file:read-text($name)
let $json := json:parse($file)
return db:add($database, document { $json }, $name)
```

Example 2: Converts a simple JSON string to XML

Query:

```
json:parse('{}')
```

Result:

```
<json objects="json"/>
```

Example 3: Converts a JSON string with simple objects and arrays

Query:

```
json:parse('{
  "title": "Talk On Travel Pool",
  "link": "http://www.flickr.com/groups/talkontravel/pool/",
  "description": "Travel and vacation photos from around the world.",
  "modified": "2009-02-02T11:10:27Z",
  "generator": "http://www.flickr.com/"
}')
```

Result:

```
<json objects="json">
  <title>Talk On Travel Pool</title>
  <link>http://www.flickr.com/groups/talkontravel/pool/</link>
  <description>Travel and vacation photos from around the world.</description>
  <modified>2009-02-02T11:10:27Z</modified>
  <generator>http://www.flickr.com/</generator>
</json>
```

Example 4: Converts a JSON string with different data types

Query:

```
json:parse('{
  "first_name": "John",
  "last_name": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "code": 10021
  },
  "phone": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": 1327724623
    }
  ]
}')
```

Result:

```
<json numbers="age code" arrays="phone" objects="json address value">
  <first__name>John</first__name>
  <last__name>Smith</last__name>
  <age>25</age>
  <address>
    <street>21 2nd Street</street>
    <city>New York</city>
    <code>10021</code>
  </address>
  <phone>
    <value>
      <type>home</type>
      <number>212 555-1234</number>
    </value>
    <value>
      <type>mobile</type>
      <number type="number">1327724623</number>
    </value>
  </phone>
</json>
```

JsonML Functions

`json:serialize-ml` and `json:parse-ml` are used to transform XML to JSON and back, using the JsonML ^[2] dialect. JsonML can be used to transform arbitrary XML documents, but namespaces, comments and processing instructions will be discarded in the transformation process. More details are found in the official JsonML documentation ^[3].

json:serialize-ml

Signatures `json:serialize-ml($input as node()) as xs:string()`

Summary Serializes the node specified by `$input` and returns the result as `xs:string` instance. XML documents can also be output in the JsonML format by setting the Serialization Option "method" to "jsonml".

Errors **BASX0016** is raised if the specified value cannot be serialized.

json:parse-ml

Signatures `json:parse-ml($input as xs:string()) as element()`

Summary Converts the JsonML ^[2] document specified by `$input` to XML, and returns the result as `element()` instance. The JSON input must conform to the JsonML specification to be successfully converted.

Errors **BASX0015** is raised if the specified input cannot be parsed as JsonML instance.

Examples

Example 1: Converts all XML documents in a database to JsonML and writes them to disk

Query:

```
for $doc in collection('json')
let $name := document-uri($doc)
let $json := json:serialize($doc)
return file:write($name, $json)
```

Example 2: Converts a simple XML fragment to the JsonML format

Query:

```
json:serialize-ml(<xml/>)
```

Result:

```
["xml"]
```

Example 3: Converts an XML document with elements and text

Query:

```
json:serialize-ml(doc('flickr.xml'))
```

flickr.xml:

```
<flickr>
  <title>Talk On Travel Pool</title>
  <link>http://www.flickr.com/groups/talkontravel/pool/</link>
  <description>Travel and vacation photos from around the world.</description>
```

```
<modified>2009-02-02T11:10:27Z</modified>
<generator>http://www.flickr.com/</generator>
</flickr>
```

Result:

```
["flickr",
  ["title",
    "Talk On Travel Pool"],
  ["link",
    "http:\\\\www.flickr.com\\groups\\talkontravel\\pool\\"],
  ["description",
    "Travel and vacation photos from around the world."],
  ["modified",
    "2009-02-02T11:10:27Z"],
  ["generator",
    "http:\\\\www.flickr.com\\"]]
```

Example 4: Converts a document with nested elements and attributes**Query:**

```
json:serialize-ml(doc('input.xml'))
```

input.xml:

```
<address id='1'>
  <!-- comments will be discarded -->
  <last_name>Smith</last_name>
  <age>25</age>
  <address xmlns='will be dropped as well'>
    <street>21 2nd Street</street>
    <city>New York</city>
    <code>10021</code>
  </address>
  <phone type='home'>212 555-1234</phone>
</address>
```

Result:

```
["address", {"id": "1"},
  ["last_name",
    "Smith"],
  ["age",
    "25"],
  ["address",
    ["street",
      "21 2nd Street"],
    ["city",
      "New York"],
    ["code",
      "10021"]],
  ["phone",
    "212 555-1234"]]
```

```
[ "phone", { "type": "home" },
  "212 555-1234" ] ]
```

Changelog

The module was introduced with Version 7.0.

References

- [1] <http://www.json.org/>
- [2] <http://jsonml.org>
- [3] <http://jsonml.org/XML>

Map Module

This XQuery Module contains functions for manipulating maps. All functions are preceded by the `map:` prefix, which is linked to the statically declared `http://www.w3.org/2005/xpath-functions/map` namespace. The following documentation is derived from an XQuery 3.0 Functions and Operators ^[6] working draft proposal written by Michael H. Kay ^[1], and is not part of the official standard yet.

A map is an additional kind of item. It comprises a collation and a set of entries. Each entry comprises a key which is an arbitrary atomic value, and an arbitrary sequence called the associated value. Within a map, no two entries have the same key, when compared using the `eq` operator under the map's collation. It is not necessary that all the keys should be mutually comparable (for example, they can include a mixture of integers and strings). Key values will never be of type `xs:untypedAtomic`, and they will never be the `xs:float` or `xs:double` value NaN.

The function call `map:get($map, $key)` can be used to retrieve the value associated with a given key.

A *map* can also be viewed as a function from keys to associated values. To achieve this, a map is also a function item. The function corresponding to the map has the signature `function($key as xs:anyAtomicType) as item()*`. Calling the function has the same effect as calling the `get` function: the expression `$map($key)` returns the same result as `map:get($map, $key)`. For example, if `$books-by-isbn` is a map whose keys are ISBNs and whose associated values are book elements, then the expression `$books-by-isbn("0470192747")` returns the book element with the given ISBN. The fact that a map is a function item allows it to be passed as an argument to higher-order functions that expect a function item as one of their arguments. As an example, the following query uses the higher-order function `fn:map($f, $seq)` to extract all bound values from a *map*:

```
let $map := map { 'foo' := 42, 'bar' := 'baz', 123 := 456 }
return fn:map($map, map:keys($map))
```

This returns some permutation of `(42, 'baz', 456)`.

Like all other values, *maps* are immutable. For example, the `map:remove` function creates a new map by removing an entry from an existing map, but the existing map is not changed by the operation.

Like sequences, *maps* have no identity. It is meaningful to compare the contents of two maps, but there is no way of asking whether they are "the same map": two maps with the same content are indistinguishable.

Because a map is a function item, functions that apply to functions also apply to maps. A map is an anonymous function, so `fn:function-name` returns the empty sequence; `fn:function-arity` always returns 1.

Maps may be compared using the `fn:deep-equal` function. The semantics for this function are extended so that when two items are compared, at any level of recursion, the items compare equal if they are both maps, if both use

the same collation, if both contain the same set of keys (compared using the `eq` operator), without regard to ordering, and if for each key that is present in both maps, the associated values are deep-equal. When comparing maps, the maps' collation is used rather than the collation supplied as an argument to the `fn:deep-equal` function.

There is no operation to atomize a map or convert it to a string. The following XQuery snippet shows how the contents of a map can be serialized:

```
let $map := map { 1:='a', 2:='b' }
return string-join(
  for $m in map:keys($map)
  return concat($m, ':', $map($m)), ', '
)
```

Some examples use the `map $week` defined as:

```
declare variable $week as map(*) := map {
  0:="Sonntag",
  1:="Montag",
  2:="Dienstag",
  3:="Mittwoch",
  4:="Donnerstag",
  5:="Freitag",
  6:="Samstag"
};
```

map:collation

Signatures `map:collation($map as map(*)) as xs:string`

Summary Returns the collation URI of the *map* supplied as *\$map*.

map:contains

Signatures `map:contains($map as map(*), $key as item()) as xs:boolean`

Summary Returns true if the *map* supplied as *\$map* contains an entry with a key equal to the supplied value of *\$key*; otherwise it returns false. The equality comparison uses the map's collation; no error occurs if the map contains keys that are not comparable with the supplied *\$key*. If the supplied key is `xs:untypedAtomic`, it is converted to `xs:string`. If the supplied key is the `xs:float` or `xs:double` value NaN, the function returns false.

Examples

- `map:contains($week, 2)` returns `true()`.
- `map:contains($week, 9)` returns `false()`.
- `map:contains(map{}, "xyz")` returns `false()`.
- `map:contains(map{ "xyz":=23 }, "xyz")` returns `true()`.

map:entry

Signatures `map:entry($key as item(), $value as item()*) as map(*)`

Summary Creates a new *map* containing a single entry. The collation of the new map is the default collation from the static context. The key of the entry in the new map is *\$key*, and its associated value is *\$value*. If the supplied key is `xs:untypedAtomic`, it is converted to `xs:string`. If the supplied key is the `xs:float` or `xs:double` value `NaN`, the supplied *\$map* is returned unchanged. The function `map:entry` is intended primarily for use in conjunction with the function `map:new`. For example, a map containing seven entries may be constructed like this:

```
map:new((
  map:entry("Su", "Sunday"),
  map:entry("Mo", "Monday"),
  map:entry("Tu", "Tuesday"),
  map:entry("We", "Wednesday"),
  map:entry("Th", "Thursday"),
  map:entry("Fr", "Friday"),
  map:entry("Sa", "Saturday")
))
```

Unlike the `map{ ... }` expression, this technique can be used to construct a map with a variable number of entries, for example:

```
map:new(for $b in //book return map:entry($b/isbn, $b))
```

Examples

- `map:entry("M", "Monday")` creates a map with the values `{ "M" := "Monday" }`.

map:get

Signatures `map:get($map as map(*), $key as item()) as item()*`

Summary Returns the value associated with a supplied key in a given map. This function attempts to find an entry within the *map* supplied as *\$map* that has a key equal to the supplied value of *\$key*. If there is such an entry, it returns the associated value; otherwise it returns an empty sequence. The equality comparison uses the map's collation; no error occurs if the map contains keys that are not comparable with the supplied *\$key*. If the supplied key is `xs:untypedAtomic`, it is converted to `xs:string`. If the supplied key is the `xs:float` or `xs:double` value `NaN`, the function returns an empty sequence. A return value of `()` from `map:get` could indicate that the key is present in the map with an associated value of `()`, or it could indicate that the key is not present in the map. The two cases can be distinguished by calling `map:contains`. Invoking the *map* as a function item has the same effect as calling `get`: that is, when *\$map* is a map, the expression `$map($K)` is equivalent to `get($map, $K)`. Similarly, the expression `get(get(get($map, 'employee'), 'name'), 'first')` can be written as `$map('employee')('name')('first')`.

Examples

- `map:get($week, 4)` returns "Donnerstag".
- `map:get($week, 9)` returns `()`. (When the key is not present, the function returns an empty sequence.).
- `map:get(map:entry(7, ()), 7)` returns `()`. (An empty sequence as the result can also signify that the key is present and the associated value is an empty sequence.).

map:keys

Signatures `map:keys` (`$map` as `map(*)`) as `xs:anyAtomicType*`

Summary Returns a sequence containing all the key values present in a map. The function takes any *map* as its `$map` argument and returns the keys that are present in the map as a sequence of atomic values, in implementation-dependent order.

Examples

- `map:keys(map{ 1:="yes", 2:="no" })` returns some permutation of `(1,2)` (*the result is in implementation-dependent order*).

map:new

Signatures `map:new` () as `map(*)`

`map:new` (`$maps` as `map(*)*`) as `map(*)`

`map:new` (`$maps` as `map(*)*`, `$coll` as `xs:string`) as `map(*)`

Summary Constructs and returns a new map. The zero-argument form of the function returns an empty *map* whose collation is the default collation in the static context. It is equivalent to calling the one-argument form of the function with an empty sequence as the value of the first argument. The one-argument form of the function returns a *map* that is formed by combining the contents of the maps supplied in the `$maps` argument. It is equivalent to calling the two-argument form of the function with the default collation from the static context as the second argument. The two-argument form of the function returns a *map* that is formed by combining the contents of the maps supplied in the `$maps` argument. The collation of the new map is the value of the `$coll` argument. The supplied maps are combined as follows:

1. There is one entry in the new map for each distinct key value present in the union of the input maps, where keys are considered distinct according to the rules of the `distinct-values` function with `$coll` as the collation.
2. The associated value for each such key is taken from the last map in the input sequence `$maps` that contains an entry with this key. If this map contains more than one entry with this key (which can happen if its collation is different from that of the new map) then it is *implementation-dependent* which of them is selected.

There is no requirement that the supplied input maps should have the same or compatible types. The type of a map (for example `map(xs:integer, xs:string)`) is descriptive of the entries it currently contains, but is not a constraint on how the map may be combined with other maps.

Examples

- `map:new()` creates an empty map.
- `map:new(())` creates an empty map.
- `map:new(map:entry(0, "no"), map:entry(1, "yes"))` creates a map with the values { 0:="no", 1:="yes" }.
- `map:new($week, map{ 7:="Unbekannt" })` creates a map with the values { 0:="Sonntag", 1:="Montag", 2:="Dienstag", 3:="Mittwoch", 4:="Donnerstag", 5:="Freitag", 6:="Samstag", 7:="Unbekannt" }.
- `map:new($week, map{ 6:="Sonnabend" })` creates a map with the values { 0:="Sonntag", 1:="Montag", 2:="Dienstag", 3:="Mittwoch", 4:="Donnerstag", 5:="Freitag", 6:="Sonnabend" }.

map:remove

Signatures `map:remove($map as map(*), $key as item()) as map(*)`

Summary Constructs a new map by removing an entry from an existing map. The collation of the new map is the same as the collation of the map supplied as `$map`. The entries in the new map correspond to the entries of `$map`, excluding any entry whose key is equal to `$key`. No failure occurs if the input map contains no entry with the supplied key; the input map is returned unchanged

Examples

- `map:remove($week, 4)` creates a map with the values { 0:="Sonntag", 1:="Montag", 2:="Dienstag", 3:="Mittwoch", 5:="Freitag", 6:="Samstag" }.
- `map:remove($week, 23)` creates a map with the values { 0:="Sonntag", 1:="Montag", 2:="Dienstag", 3:="Mittwoch", 4:="Donnerstag", 5:="Freitag", 6:="Samstag" }.

map:size

Signatures `map:size($map as map(*)) as xs:integer`

Summary Returns a the number of entries in the supplied map. The function takes any *map* as its `$map` argument and returns the number of entries that are present in the map.

Examples

- `map:size(map:new())` returns 0.
- `map:size(map{ "true":=1, "false":=0 })` returns 2.

References

[1] [http://en.wikipedia.org/wiki/Michael_Kay_\(software_engineer\)](http://en.wikipedia.org/wiki/Michael_Kay_(software_engineer))

Math Module

The math XQuery Module defines functions to perform mathematical operations, such as `pi`, `asin` and `acos`. All functions are preceded by the `math:` prefix, which is linked to the statically declared <http://www.w3.org/2005/xpath-functions/math> namespace. Some of the functions have also been specified in the Functions and Operators Specification^[6] of the upcoming XQuery 3.0 Recommendation.

math:pi

Signatures `math:pi()` as `xs:double`

Summary Returns the `xs:double` value of the mathematical constant π whose lexical representation is 3.141592653589793.

Examples

- `2*math:pi()` returns 6.283185307179586e0.
- `60 * (math:pi() div 180)` converts an angle of 60 degrees to radians.

math:e

Signatures `math:e()` as `xs:double`

Summary Returns the `xs:double` value of the mathematical constant e whose lexical representation is 2.718281828459045.

Examples • `5*math:e()` returns 13.591409142295225.

math:sqrt

Signatures `math:sqrt($arg as xs:double?)` as `xs:double?`

Summary Returns the square root of `$arg`.

If `$arg` is the empty sequence, the empty sequence is returned.

Otherwise the result is the `xs:double` value of the mathematical square root of `$arg`.

math:sin

Signatures `math:sin($arg as xs:double?)` as `xs:double?`

Summary Returns the sine of the `$arg`, expressed in radians.

If `$arg` is the empty sequence, the empty sequence is returned.

Otherwise the result is the sine of `$arg`, treated as an angle in radians.

math:cos

Signatures `math:cos($arg as xs:double?)` as `xs:double?`

Summary Returns the cosine of `$arg`, expressed in radians.

If `$arg` is the empty sequence, the empty sequence is returned.

Otherwise the result is the cosine of `$arg`, treated as an angle in radians.

math:tan

Signatures `math:tan($ as xs:double?)` as `xs:double?`

Summary Returns the tangent of `$arg`, expressed in radians.

If `$arg` is the empty sequence, the empty sequence is returned.

Otherwise the result is the tangent of `$arg`, treated as an angle in radians.

math:asin

Signatures `math:asin`(\$arg as xs:double?) as xs:double?

Summary Returns the arc sine of \$arg.
If \$arg is the empty sequence, the empty sequence is returned.
Otherwise the result is the arc sine of \$arg, returned as an angle in radians in the range $-\pi/2$ to $+\pi/2$.

math:acos

Signatures `math:acos`(\$arg as xs:double?) as xs:double?

Summary Returns the arc cosine of \$arg.
If \$arg is the empty sequence, the empty sequence is returned.
Otherwise the result is the arc cosine of \$arg, returned as an angle in radians in the range 0 to $+\pi$.

math:atan

Signatures `math:atan`(\$arg as xs:double?) as xs:double?

Summary Returns the arc tangent of \$arg.
If \$arg is the empty sequence, the empty sequence is returned.
Otherwise the result is the arc tangent of \$arg, returned as an angle in radians in the range $-\pi/2$ to $+\pi/2$.

math:atan2

Signatures `math:atan2`(\$arg1 as xs:double?, \$arg2 as xs:double) as xs:double?

Summary Returns the arc tangent of \$arg1 divided by \$arg2, the result being in the range $-\pi/2$ to $+\pi/2$ radians.
If \$arg1 is the empty sequence, the empty sequence is returned.
Otherwise the result is the arc tangent of \$arg1 divided by \$arg2, returned as an angle in radians in the range $-\pi$ to $+\pi$.

math:pow

Signatures `math:pow`(\$arg1 as xs:double?, \$arg2 as xs:double) as xs:double?

Summary Returns \$arg1 raised to the power of \$arg2.
If \$arg1 is the empty sequence, the empty sequence is returned.
Otherwise the result is the \$arg1 raised to the power of \$arg2.

Examples • `math:pow(2, 3)` returns 8.

math:exp

Signatures `math:exp($arg as xs:double?) as xs:double?`

Summary Returns e raised to the power of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the value of e raised to the power of `$arg`.

Examples • `math:exp(1)` returns e .

math:log

Signatures `math:log($arg as xs:double?) as xs:double?`

Summary Returns the natural logarithm of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the natural logarithm (base e) of `$arg`.

Examples • `math:log(math:e())` returns 1.

math:log10

Signatures `math:log10($arg as xs:double?) as xs:double?`

Summary Returns the base 10 logarithm of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the base 10 logarithm of `$arg`.

Examples • `math:log(100)` returns 2.

math:random

Signatures `math:random() as xs:double?`

Summary Returns a random `xs:double` value between 0.0 and 1.0.

math:sinh

Signatures `math:sinh($arg as xs:double?) as xs:double?`

Summary Returns the hyperbolic sine of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the hyperbolic sine of `$arg`.

Examples • `math:sinh(0)` returns 0.

math:cosh

Signatures `math:cosh($arg as xs:double?) as xs:double?`

Summary Returns the hyperbolic cosine of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the hyperbolic cosine of `$arg`.

Examples • `math:cosh(0)` returns 1.

math:tanh

Signatures `math:tanh($arg as xs:double?) as xs:double?`

Summary Returns the hyperbolic tangent of `$arg`.
If `$arg` is the empty sequence, the empty sequence is returned.
Otherwise the result is the hyperbolic tangent of `$arg`.

Examples • `math:tanh(100)` returns 1.

Repository Module

This XQuery Module contains functions for installing, listing and deleting modules contained in the Repository. All functions are preceded by the `repo:` prefix which is linked to the namespace <http://basex.org/modules/repo>.

Functions

repo:install

Signatures `repo:install($path as xs:string) as empty-sequence()`

Summary Installs a package, or (since [Version 7.2.1](#)), replaces an existing package. The parameter `$path` indicates the path to the package.

Errors

- PACK0001** is raised if the package does not exist.
- PACK0002** is raised if a package uses an invalid namespace URI.
- PACK0003** is raised if the package to be installed requires a package which is still not installed.
- PACK0004** is raised if the package descriptor is invalid.
- PACK0005** is raised if the module contained in the package to be installed is already installed as part of another package.
- PACK0006** is raised if the package cannot be parsed.
- PACK0009** is raised if the package version is not supported.
- PACK0010** is raised if the package contains an invalid JAR descriptor.
- PACK0011** is raised if the package contains a JAR descriptor but it cannot be read.

repo:delete

Signatures `repo:delete($pkg as xs:string) as empty-sequence()`

Summary Deletes a package. The parameter `$pkg` indicates either the package name as specified in the package descriptor or (since [Version 7.2.1](#)) the name, suffixed with a hyphen and the package version.

Errors **PACK0007** is raised if the package cannot be deleted.
PACK0008 is raised if another package depends on the package to be deleted.

repo:list

Updated in [Version 7.2](#):

Signatures `repo:list() as element(package)*`

Summary Lists the names and versions of all currently installed packages.

Changelog

Version 7.2.1

- Updated: `repo:install`: existing packages will be replaced
- Updated: `repo:delete`: remove specific version of a package

Version 7.2

- Updated: `repo:list` now returns nodes

The module was introduced with [Version 7.1](#).

SQL Module

This XQuery Module contains functions to access relational databases from XQuery using SQL. With this module, you can execute query, update and prepared statements, and the result sets are returned as sequences of XML elements representing tuples. Each element has children representing the columns returned by the SQL statement. All functions dealing with access to relational databases use the `sql` prefix, which is linked to the statically declared `http://basex.org/modules/sql` namespace.

Functions

sql:init

Signatures `sql:init($class as xs:string) as empty-sequence()`

Summary This function initializes a JDBC driver specified via `$class`. This step might be superfluous if the SQL database is not embedded.

Errors **FOSQ0007** is raised if the specified driver class is not found.

sql:connect

Signatures `sql:connect($url as xs:string) as xs:integer`

`sql:connect($url as xs:string, $user as xs:string, $password as xs:string) as xs:integer`

`sql:connect($url as xs:string, $user as xs:string, $password as xs:string, $options as item()) as xs:integer`

Summary This function establishes a connection to a relational database. As a result a connection handle is returned. The parameter `$url` is the URL of the database and shall be of the form: `jdbc:<driver name>:[//<server>[/<database>]`. If the parameters `$user` and `$password` are specified, they are used as credentials for connecting to the database. The parameter `$options` can be used to set connection options, e.g. auto-commit mode. It can be specified as:

- `element(sql:options): <sql:options/>` must be used as root element, and the options are specified as child nodes, with the element name representing the key and the text node representing the value:

```
<sql:options>
  <sql:autocommit>true</sql:autocommit>
  ...
</sql:options>
```

- `map` structure: all options can be directly represented as key/value pairs:


```
map { "autocommit" := "true", ... }
```

Errors **FOSQ0001** is raised if an SQL exception occurs, e.g. missing JDBC driver or not existing relation.

sql:execute

Once a connection is established, the returned connection handle can be used to execute queries on the database. Our SQL module supports both direct queries and prepared statements.

Signatures `sql:execute`(\$connection as xs:integer, \$item as item()) as element()*

Summary This function executes a query, update or prepared statement. The parameter `$id` specifies either a connection handle or a prepared statement handle. The parameter `$item` is either a string representing an SQL statement or an element `<sql:parameters/>` representing the parameters for a prepared statement along with their types and values. In case of the latter, the following schema shall be used:

```
element sql:parameters {
  element sql:parameter {
    attribute type { "int"|"string"|"boolean"|"date"|"double"|"float"|"short"|"time"|"timestamp" },
    attribute null { "true"|"false" }?,
    text }+ }?
```

Errors

- FOSQ0001** is raised if an SQL exception occurs, e.g. not existing relation is retrieved.
- FOSQ0002** is raised if a wrong connection handle or prepared statement handle is passed.
- FOSQ0003** is raised if the number of `<sql:parameter/>` elements in `<sql:parameters/>` differs from the number of placeholders in the prepared statement.
- FOSQ0004** is raised if the type of a parameter for a prepared statement is not specified.
- FOSQ0005** is raised if an attribute different from `type` and `null` is set for a `<sql:parameter/>` element.
- FOSQ0006** is raised if a parameter is from type `date`, `time` or `timestamp` and its value is in an invalid format.

sql:prepare

Signatures `sql:prepare`(\$connection as xs:integer, \$statement as xs:string) as xs:integer

Summary This function prepares a statement and returns a handle to it. The parameter `$connection` indicates the connection handle to be used. The parameter `$statement` is a string representing an SQL statement with one or more '?' placeholders. If the value of a field has to be set to NULL, then the attribute `null` of the element `<sql:parameter/>` has to be `true`.

Errors

- FOSQ0001** is raised if an SQL exception occurs.
- FOSQ0002** is raised if a wrong connection handle is passed.

sql:commit

Signatures `sql:commit`(\$connection as xs:integer) as xs:element?

Summary This function commits the changes made to a relational database. `$connection` specifies the connection handle.

Errors

- FOSQ0001** is raised if an SQL exception occurs.
- FOSQ0002** is raised if a wrong connection handle is passed.

sql:rollback

Signatures `sql:rollback`(\$connection as xs:integer) as xs:element?

Summary This function rolls back the changes made to a relational database. `$connection` specifies the connection handle.

Errors

- FOSQ0001** is raised if an SQL exception occurs.
- FOSQ0002** is raised if a wrong connection handle is passed.

sql:close

Signatures `sql:close($connection as xs:integer) as xs:element?`

Summary This function closes a connection to a relational database. `$connection` specifies the connection handle.

Errors **FOSQ0001** is raised if an SQL exception occurs.
FOSQ0002 is raised if a wrong connection handle is passed.

Examples

Direct queries

A simple select statement can be executed on the following way:

```
let $conn := sql:connect("jdbc:postgresql://localhost:5432/coffeehouse")
return sql:execute($conn, "SELECT * FROM coffees WHERE price < 10")
```

The result will look like:

```
<sql:row xmlns:sql="http://basex.org/modules/sql">
  <sql:column name="cof_name">French_Roast</sql:column>
  <sql:column name="sup_id">49</sql:column>
  <sql:column name="price">9.5</sql:column>
  <sql:column name="sales">15</sql:column>
  <sql:column name="total">30</sql:column>
</sql:row>
<sql:row xmlns:sql="http://basex.org/modules/sql">
  <sql:column name="cof_name">French_Roast_Decaf</sql:column>
  <sql:column name="sup_id">49</sql:column>
  <sql:column name="price">7.5</sql:column>
  <sql:column name="sales">10</sql:column>
  <sql:column name="total">14</sql:column>
</sql:row>
<sql:row xmlns:sql="http://basex.org/modules/sql">
  <sql:column name="cof_name">Colombian_Decaf</sql:column>
  <sql:column name="sup_id">101</sql:column>
  <sql:column name="price">8.75</sql:column>
  <sql:column name="sales">6</sql:column>
  <sql:column name="total">12</sql:column>
  <sql:column name="date">2010-10-10 13:56:11.0</sql:column>
</sql:row>
```

Prepared Statements

A prepared select statement can be executed in the following way:

```
(: Establish a connection :)
let $conn := sql:connect("jdbc:postgresql://localhost:5432/coffeehouse")
(: Obtain a handle to a prepared statement :)
let $prep := sql:prepare($conn, "SELECT * FROM coffees WHERE price < ? AND cof_name = ?")
(: Values and types of prepared statement parameters :)
let $params := <sql:parameters>
    <sql:parameter type='double'>10</sql:parameter>
    <sql:parameter type='string'>French_Roast</sql:parameter>
</sql:parameters>
(: Execute prepared statement :)
return sql:execute($prep, $params)
```

Changelog

The module was introduced with Version 7.0.

Utility Module

This XQuery Module contains auxiliary functions that can be used to perform data conversions and test and profile code snippets. All functions are preceded by the `util:` prefix, which is linked to the statically declared `http://basex.org/modules/util` namespace.

Conversion

`util:format`

Signatures `util:format`(\$format as xs:string, \$item1 as item(), ...) as xs:string

Summary Returns a formatted string. `$item1` and all following items are applied to the `$format` string, according to Java's `printf` syntax ^[1].

Examples

- `util:format("%b", true())` returns `true`.
- `util:format("%06d", 256)` returns `000256`.
- `util:format("%e", 1234.5678)` returns `1.234568e+03`.

`util:integer-to-base`

Signatures `util:integer-to-base`(\$num as xs:integer, \$base as xs:integer) as xs:string

Summary Converts \$num to base \$base, interpreting it as a 64-bit unsigned integer.
The first \$base elements of the sequence '0', .., '9', 'a', .., 'z' are used as digits.
Valid bases are 2, .., 36.

Examples

- `util:integer-to-base(-1, 16)` returns the hexadecimal string 'ffffffffffffffff'.
- `util:integer-to-base(22, 5)` returns '42'.

util:integer-from-base

Signatures `util:integer-from-base`(\$str as xs:string, \$base as xs:integer) as xs:integer

Summary Decodes an xs:integer from \$str, assuming that it's encoded in base \$base.
The first \$base elements of the sequence '0', .., '9', 'a', .., 'z' are allowed as digits, case doesn't matter.
Valid bases are 2, .., 36.
If \$str contains more than 64 bits of information, the result is truncated arbitrarily.

Examples

- `util:integer-from-base('ffffffffffffffff', 16)` returns -1.
- `util:integer-from-base('CAFEBABE', 16)` returns 3405691582.
- `util:integer-from-base('42', 5)` returns 22.
- `util:integer-from-base(util:integer-to-base(123, 7), 7)` returns 123.

util:to-bytes

Signatures `util:to-bytes`(\$bin as xs:binary) as xs:byte*

Summary Extracts the bytes from the given binary data \$bin.

Examples

- `util:to-bytes(xs:base64Binary('QmFzZVggaXMgY29vbA=='))` returns the sequence (66, 97, 115, 101, 88, 32, 105, 115, 32, 99, 111, 111, 108).
- `util:to-bytes(xs:hexBinary("4261736558"))` returns the sequence (66 97 115 101 88).

util:to-string

Signatures `util:to-string`(\$bytes as xs:binary, \$encoding as xs:string) as xs:string

Summary Converts the specified bytes to a string, using the optional \$encoding.

Examples

- `util:to-string(xs:hexBinary('48656c6c6f576f726c64'))` returns the string HelloWorld.

Reflection, Introspection

util:eval

Signatures `util:eval($expr as xs:string) as item()*`

Summary Evaluates `$expr` as XQuery expression at runtime and returns the resulting items.

Examples • `util:eval("1+3")` returns 4.

util:run

Signatures `util:run($input as xs:string) as item()*`

Summary Opens `$input` as file, evaluates it as XQuery expression at runtime, and returns the resulting items.

util:path

Signatures `util:path() as xs:string?`

Summary Returns the full path to the file containing the executed query. An empty sequence is returned if no such file exists, or if the filename is not known to the query processor.

util:type

Added in [Version 7.2](#).

Signatures `util:type($expr as item()) as item()*`

Summary Similar to `fn:trace($expr, $msg)`, but instead of a user-defined message, it emits the compile-time type and estimated result size of its argument.

Profiling

util:mem

In [Version 7.2](#), the `$label` argument was added to the function:

Signatures `util:mem($expr as item()) as item()*`

`util:mem($expr as item(), $cache as xs:boolean) as item()*`

`util:mem($expr as item(), $cache as xs:boolean, $label as xs:string) as item()*`

Summary Measures the memory allocated by evaluating `$expr` and sends it to standard error or, if the GUI is used, to the Info View.

If `$cache` is set to `true()`, the result will be temporarily cached. This way, a potential iterative execution of the expression (which often yields different memory usage) is blocked.

A third, optional argument `$label` may be specified to tag the profiling result.

Examples • `util:mb("1 to 100000")` may output 0 Bytes.
• `util:mb("1 to 100000", true())` may output 26.678 mb.

util:time

In [Version 7.2](#), the `$label` argument was added to the function:

Signatures `util:time($expr as item()) as item()*`
`util:time($expr as item(), $cache as xs:boolean) as item()*`
`util:time($expr as item(), $cache as xs:boolean, $label as xs:string) as item()*`

Summary Measures the time needed to evaluate `$expr` and sends it to standard error or, if the GUI is used, to the Info View. If `$cache` is set to `true()`, the result will be temporarily cached. This way, a potential iterative execution of the expression (which often yields different memory usage) is blocked. A third, optional argument `$label` may be specified to tag the profiling result.

Examples

- `util:time("1 to 100000")` may output 25.69 ms.
- `util:time("1 to 100000", true())` may output 208.12 ms.

util:sleep

Introduced with [Version 7.2.1](#):

Signatures `util:sleep($ms as xs:integer) as empty-sequence()*`

Summary Sleeps for the specified number of milliseconds.

Cryptography

util:md5

Signatures `util:md5($str as xs:string) as xs:hexBinary`

Summary Calculates the MD5 hash of the given string `$str`.

Examples

- `util:md5("")` returns 'D41D8CD98F00B204E9800998ECF8427E'.
- `util:md5('BaseX')` returns '0D65185C9E296311C0A2200179E479A2'.

util:sha1

Signatures `util:sha1($str as xs:string) as xs:hexBinary`

Summary Calculates the SHA1 hash of the given string `$str`.

Examples

- `util:sha1("")` returns 'DA39A3EE5E6B4B0D3255BFEEF95601890AFD80709'.
- `util:sha1("BaseX")` returns '3AD5958F0F27D5AFFDCA2957560F121D0597A4ED'.

util:crc32

Signatures `util:crc32($str as xs:string) as xs:hexBinary`

Summary Calculates the CRC32 check sum of the given string `$str`.

Examples

- `util:crc32("")` returns '00000000'.
- `util:crc32("BaseX")` returns '4C06FC7F'.

util:uuid

Signatures `util:uuid() as xs:string`

Summary Creates a random universally unique identifier (UUID), represented as 128-bit value.

Examples

- `util:uuid() ne util:uuid()` will (most probably) return the boolean value `true`.

Changelog

Version 7.2.1

- Updated: `util:sleep`

Version 7.2

- Updated: `util:time`, `util:mem`, `util:type`

Version 7.1

- Added: `util:path`, `util:time`, `util:mem`
- Removed: `util:mb`, `util:ms`

Version 7.0

- Added: `util:to-string`, `util:uuid`

References

- [1] <http://download.oracle.com/javase/1.5.0/docs/api/java/util/Formatter.html#syntax>

XSLT Module

This XQuery Module contains functions and variables to perform XSLT transformations. All functions are preceded by the `xslt:` prefix, which is linked to the statically declared `http://basex.org/modules/xslt` namespace.

By default, this module uses Java's XSLT 1.0 Xalan implementation to transform documents. XSLT 2.0 is used instead if Version 9.x of the Saxon XSLT Processor ^[1] (`saxon9he.jar`, `saxon9pe.jar`, `saxon9ee.jar`) is found in the classpath. A custom transformer can be specified by overwriting the system property `javax.xml.transform.TransformerFactory`, as shown in the following Java example:

```
System.setProperty("javax.xml.transform.TransformerFactory", "org.custom.xslt.TransformerFactoryImpl");
Context ctx = new Context();
String result = new XQuery("xslt:transform('...', '...').execute(ctx);
...
ctx.close();
```

\$xslt:processor

Signatures `$xslt:processor` as `xs:string`

Summary This variable contains the name of the applied XSLT processor, or the path to a custom implementation (currently: "Java", "Saxon EE", "Saxon PE", or "Saxon HE").

\$xslt:version

Signatures `$xslt:version` as `xs:string`

Summary This variable contains the supported XSLT version (currently: "1.0" or "2.0"). "Unknown" is returned if a custom implementation was chosen.

xslt:transform

Signatures `xslt:transform($input as item(), $stylesheet as item()) as node()`
`xslt:transform($input as item(), $stylesheet as item(), $params as item()) as node()`

Summary Transforms the document specified by `$input`, using the XSLT template specified by `$stylesheet`, and returns the result as `node()` instance. `$input` and `$stylesheet` can be specified as

- `xs:string`, containing the path to the document,
- `xs:string`, containing the document in its string representation, or
- `node()`, containing the actual document.

The `$params` argument can be used to bind variables to a stylesheet. It can be specified as

- `element(xslt:parameters):<xslt:parameters/>` must be used as root element, and the parameters are specified as child nodes, with the element name representing the key and the text node representing the value:

```
<xslt:parameters>
  <xslt:key1>value1</xslt:key1>
  ...
</xslt:parameters>
```

- `map` structure: all parameters can be directly represented as key/value pairs:

```
map { "key1" := "value1", ... }
```

This variant is more compact, and it facilitates the binding of arbitrary data types. Note that only strings are supported when using Saxon (XSLT 2.0). Next, the map structures are not part of the XQuery language yet, as the standardization is still in progress.

Examples

Example 1: Basic XSL transformation with dummy document and without parameters

Query:

```
xslt:transform(<dummy/>, 'basic.xslt')
```

basic.xslt

```
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:template match="/">
    <result/>
  </xsl:template>
</xsl:stylesheet>
```

Result:

```
<result/>
```

Example 2: XSLT transformation of an input document

Query:

```
let $in :=
  <books>
    <book>
      <title>XSLT Programmer's Reference</title>
      <author>Michael H. Kay</author>
    </book>
    <book>
      <title>XSLT</title>
      <author>Doug Tidwell</author>
      <author>Simon St. Laurent</author>
      <author>Robert Romano</author>
    </book>
  </books>
let $style :=
  <html xsl:version='2.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform' xmlns='http://www.w3.org/1999/xhtml'>
    <body>
      <h1>Books</h1>
      <ul>
        <xsl:for-each select='books/book'>
          <li>
            <b><xsl:apply-templates select='title'/></b>: <xsl:value-of select='string-join(author, ", ")/>
          </li>
        </xsl:for-each>
      </ul>
    </body>
  </html>
return xslt:transform($in, $style)
```

Result:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <body>
    <h1>Books</h1>
    <ul>
      <li><b>XSLT Programmer's Reference</b>: Michael H. Kay</li>
      <li><b>XSLT</b>: Doug Tidwell, Simon St. Laurent, Robert Romano</li>
    </ul>
  </body>
</html>
```

Example 3: Assigning a variable to an XSLT stylesheet

Query:

```
let $in := <dummy/>
let $style := doc('variable.xsl')
return (
  xslt:transform($in, $style, <xslt:parameters><xslt:v>1</xslt:v></xslt:parameters>),
  xslt:transform($in, $style, map { "v" := 1 })
)
```

variable.xsl

```
<xsl:stylesheet version='1.0'
  xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:param name='v'/>
  <xsl:template match='/'>
    <v><xsl:value-of select='$v'/></v>
  </xsl:template>
</xsl:stylesheet>
```

Result:

```
<v>1</v>
<v>1</v>
```

References

- [1] <http://www.saxonica.com/>

ZIP Module

This XQuery Module contains functions to handle ZIP archives. The contents of ZIP files can be extracted and listed, and new archives can be created. All functions are preceded by the `zip:` prefix, which is linked to the statically declared `http://expath.org/ns/zip` namespace. The module is based on the EXPath ZIP Module [5].

Another page in this Wiki demonstrates how to modify Word documents with the ZIP module.

zip:binary-entry

Signatures `zip:binary-entry($uri as xs:string, $path as xs:string) as xs:base64Binary`

Summary Extracts the binary file at `$path` within the ZIP file located at `$uri` and returns it as an `xs:base64Binary` item.

Errors **FOZP0001** is raised if the specified path does not exist.
FOZP0003 is raised if the operation fails for some other reason.

zip:text-entry

Signatures `zip:text-entry($uri as xs:string, $path as xs:string) as xs:string`

`zip:text-entry($uri as xs:string, $path as xs:string, $encoding as xs:string) as xs:string`

Summary Extracts the text file at `$path` within the ZIP file located at `$uri` and returns it as an `xs:string` item.
 An optional encoding can be specified via `$encoding`.

Errors **FOZP0001** is raised if the specified path does not exist.
FOZP0003 is raised if the operation fails for some other reason.

zip:xml-entry

Signatures `zip:xml-entry($uri as xs:string, $path as xs:string) as document-node()`

Summary Extracts the XML file at `$path` within the ZIP file located at `$uri` and returns it as a document node.

Errors **FODC0006** is raised if the addressed file is not well-formed.
FOZP0001 is raised if the specified path does not exist.
FOZP0003 is raised if the operation fails for some other reason.

zip:html-entry

Signatures `zip:html-entry`(\$uri as xs:string, \$path as xs:string) as document-node()

Summary Extracts the HTML file at \$path within the ZIP file located at \$uri and returns it as a document node. The file is converted to XML first if Tagsoup^[2] is found in the classpath.

Errors **FODC0006** is raised if the addressed file is not well-formed, or cannot be converted to correct XML.
FOZP0001 is raised if the specified path does not exist.
FOZP0003 is raised if the operation fails for some other reason.

zip:entries

Signatures `zip:entries`(\$uri as xs:string) as element(zip:file)

Summary Generates an ZIP XML Representation^[1] of the hierarchical structure of the ZIP file located at \$uri and returns it as an element node. The file contents are not returned by this function.

Errors **FOZP0001** is raised if the specified path does not exist.
FOZP0003 is raised if the operation fails for some other reason.

Examples • If the ZIP archive archive.zip is empty, `zip:entries('archive.zip')` returns `<zip:file xmlns:zip="http://expath.org/ns/zip" href="archive.zip"/>`.

zip:zip-file

Signatures `zip:zip-file`(\$zip as element(zip:file)) as empty-sequence()

Summary Creates a new ZIP archive with the characteristics described by \$zip, the ZIP XML Representation^[1].

Errors **FOZP0001** is raised if an addressed file does not exist.
FOZP0002 is raised if entries in the ZIP archive description are unknown, missing, or invalid.
FOZP0003 is raised if the operation fails for some other reason.
Serialization Errors are raised if an inlined XML fragment cannot be successfully serialized.

Examples • The following function creates a file archive.zip with the file file.txt inside:

```
zip:zip-file(
  <file xmlns="http://expath.org/ns/zip" href="archive.zip">
    <entry src="file.txt"/>
  </file>)
```

• The following function creates a file archive.zip. It contains one file readme with the content "thanks":

```
zip:zip-file(
  <file xmlns="http://expath.org/ns/zip" href="archive.zip">
    <entry name="readme">thanks</entry>
  </file>)
```

zip:update-entries

Signatures `zip:update-entries` (\$zip as element(zip:file), \$output as xs:string) as empty-sequence()

Summary Updates an existing ZIP archive or creates a modified copy, based on the characteristics described by \$zip, the ZIP XML Representation ^[1]. The \$output argument is the URI where the modified ZIP file is copied to.

Errors **FOZP0001** is raised if an addressed file does not exist.
FOZP0002 is raised if entries in the ZIP archive description are unknown, missing, or invalid.
FOZP0003 is raised if the operation fails for some other reason.
Serialization Errors are raised if an inlined XML fragment cannot be successfully serialized.

Examples • The following function creates a copy new.zip of the existing archive.zip file:

```
zip:update-entries(zip:entries('archive.zip'), 'new.zip')
```

• The following function deletes all PNG files from archive.zip:

```
declare namespace zip = "http://expath.org/ns/zip";
copy $doc := zip:entries('archive.zip')
modify delete node $doc//zip:entry[ends-with(lower-case(@name), '.png')]
return zip:update-entries($doc, 'archive.zip')
```

References

[1] <http://expath.org/spec/zip#spec-file-handling-elements-sect>

ZIP Module: Word Documents

Contents of Word and Open Office documents can be modified with the help of the ZIP Module, as demonstrated in the following example:

Example

1. Create a new Word document writing "HELLO WORLD!" in it and save it as "HelloWorld.docx".
2. Get archive structure of "HelloWorld.docx" using the following query:

```
zip:entries("HelloWorld.docx")
```

3. Look at the XML representation and locate "document.xml"
4. Get the XML representation of "document.xml" with the following query:

```
zip:xml-entry("HelloWorld.docx", "word/document.xml")
```

5. Look at the XML representation of "document.xml"
6. Locate the element containing text "HELLO WORLD!" in "document.xml":

```
for $i in zip:xml-entry("HelloWorld.docx", "word/document.xml")/*
where contains($i/text(), "HELLO WORLD!")
return $i
```

7. Modify the contents of the element:

```
declare variable $input := "HelloWorld.docx";
declare variable $output := "HelloUniverse.docx";

let $modified :=
```

```
copy $c := zip:xml-entry($input, "word/document.xml")
modify replace value of node $c//*[text() = "HELLO WORLD!"] with "HELLO UNIVERSE!"
return $c
let $target :=
  copy $c := zip:entries($input)
  modify
    insert node $modified into $c//*[@name = "document.xml"]
  return $c
return zip:update-entries($target, $output)
```

8. Open the newly created Word document named "HelloUniverse.docx"

9. You can also modify the font of the Word document:

```
declare variable $input := "Arial.docx";
declare variable $output := "Times.docx";

let $modified :=
  copy $c := zip:xml-entry($input, "word/document.xml")
  modify for $i in $c//*[@data() = "Arial"]
    return replace value of node $i with "Times New Roman"
  return $c
let $target :=
  copy $c := zip:entries($input)
  modify insert node $modified into $c//*[@name = "document.xml"]
  return $c
return zip:update-entries($target, $output)
```

10. Open the new created Word document named "HelloWorldTimes.docx"

Developing

Developing

This page is one of the Main Sections of the documentation. It provides useful information for developers. Here you can find information on various alternatives to integrate BaseX into your own project.

Integrate & Contribute

- Integrate: Compile and run BaseX from within Eclipse
- Git: Learn how to work with Git
- Maven: Embed BaseX into your own projects
- Releases: Official releases, snapshots, old versions
- Translations: Contribute a new translation to BaseX

JavaDoc

The project's JavaDoc ^[1] can be explored online.

HTTP Services

- [Version 7.2](#): RESTXQ: Write web services with XQuery
- REST: Access and update databases via HTTP requests
- WebDAV: Access databases from your filesystem

Client APIs

- Clients: Communicate with BaseX using C#, PHP, Python, Perl, C, ...
- [Version 7.2.1](#): XQJ API ^[2], implemented by Charles Foster
- Java Examples: Code examples for developing with BaseX

References

[1] <http://docs.basex.org/javadoc>

[2] <http://xqj.net/basex>

Integrate

This page is part of the Developer Section. It describes how to get the BaseX sources compiled and running on your system.

Prerequisites

- BaseX is being developed with the Eclipse ^[1] environment. Other IDEs are used as well in our community, but are not supported by our team.
- The EGit ^[1] plugin can be used to check out the latest sources from our repository within Eclipse.
- The m2eclipse ^[2] plugin is required to work with packages other than the main project; it adds Maven support to Eclipse.
- Additional coding guidelines are defined via Checkstyle and can be integrated with the eclipse-cs ^[3] plugin.
- Other Eclipse plugins we frequently use are FindBugs ^[4] to analyze Java byte code, and Core Tools ^[5] to find unreferenced members.

You may as well use the standalone version of Maven to compile and run the project.

Check Out

To get some help on how to check out BaseX and its sub projects from the GitHub Repositories ^[6], and how to optionally use BaseX on command line, please have a look at our Git Tutorial.

The following repositories are available:

1. basex ^[7] is the main project
2. basex-api ^[8] contains the BaseX APIs (XML:DB, bindings in other languages) and HTTP Services (REST, RESTXQ, WebDAV)
3. basex-examples ^[9] includes some examples code for BaseX
4. basex-tests ^[10] contains several correctness and stress tests

Start in Eclipse

1. Press *Run* → *Run...*
2. Create a new "Java Application" launch configuration
3. Select "basex" as "Project"
4. Choose a "Main class" (e.g., `org.basex.BaseXGUI` for the graphical user interface)
5. Launch the project via *Run*

References

- [1] <http://www.eclipse.org/egit/>
 - [2] <http://m2eclipse.sonatype.org>
 - [3] <http://eclipse-cs.sourceforge.net>
 - [4] <http://findbugs.sourceforge.net/>
 - [5] <http://www.eclipse.org/eclipse/platform-core/downloads.php>
 - [6] <https://github.com/BaseXdb>
 - [7] <https://github.com/BaseXdb/basex>
 - [8] <https://github.com/BaseXdb/basex-api>
 - [9] <https://github.com/BaseXdb/basex-examples>
 - [10] <https://github.com/BaseXdb/basex-tests>
-

Git

This page is part of the Developer Section. It describes how to use git^[1] to manage the BaseX sources.

Using Git to contribute to BaseX

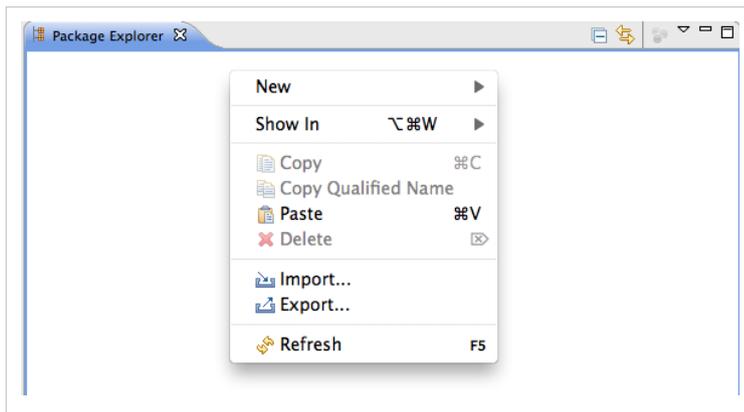
Our team uses git and GitHub^[2] to manage the source code. All team members have read+write access to the repository, and external contributors are welcome to fork the project.

Git makes it easy to retain a full copy of the repository for yourself. To get started and running, simply *fork* BaseX. If forking sounds unfamiliar to you, we suggest to check out the git introduction below.

You can then build BaseX with Maven. Using Eclipse is optional.

Using Git & Eclipse

1. (Optional) Head over to <https://github.com/BaseXdb> and create an account
2. Fork BaseX, so you have a version on your own
3. Make yourself familiar with git (see the end of this page)
4. Open Eclipse
5. Install egit (Eclipse: *Help* → *Marketplace* → Search for *egit* **or** get it from <http://www.eclipse.org/egit/>)

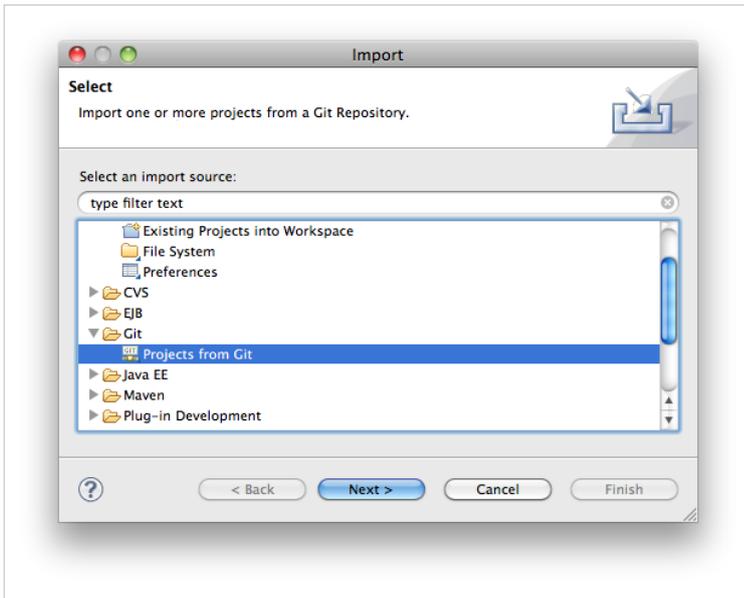


Clone

- In the **Package Explorer** to the left use right-click and choose Import...
- Select "**Projects from Git**" and click Next >
- Click "**Clone...**" to create a local copy of the remote repository. This copy will include the full project history
- Copy & Paste the github URI in the Location field. If you want to use SSH

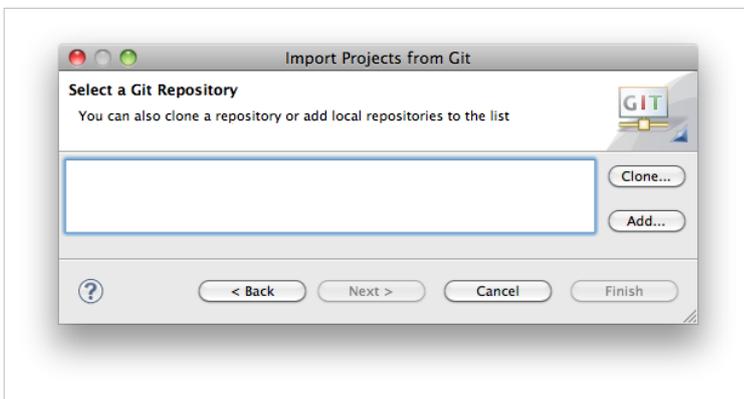
make sure you provided GitHub with your public key to allow write-access. If in doubt use the HTTPS URI and authenticate yourself with your GitHub credentials.

- Select the master branch (or arbitrary branches you like)
- Now choose a location where the local repository is stored: Create `<workspace>/repos/BaseX` and click "**Finish**".



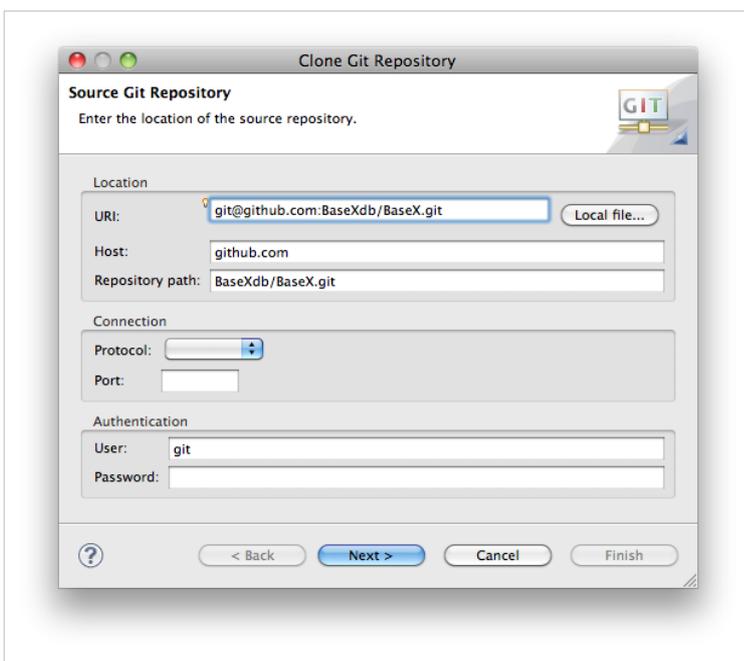
Create the project

- Select our newly cloned repository and click Next
- Select **"Import Existing Projects"** and depending on your Eclipse version enable automatic sharing. More recent versions will not offer this feature as sharing is enabled by default.
- Click next to select the Project to import
- Check "basex" to checkout and click finish
- You are now ready to contribute.



EGit & SSH

EGit uses the JSch ^[3] library which is, however, reported ^[4] to have problems with RSA SSH keys in linux and possibly other platforms. A solution would be to use the variable GIT_SSH and assign it a path to the native SSH executable. According to this ^[5] change in EGit, the plugin will try to use a native SSH implementation instead of JSch (this, however, may not always work either :().



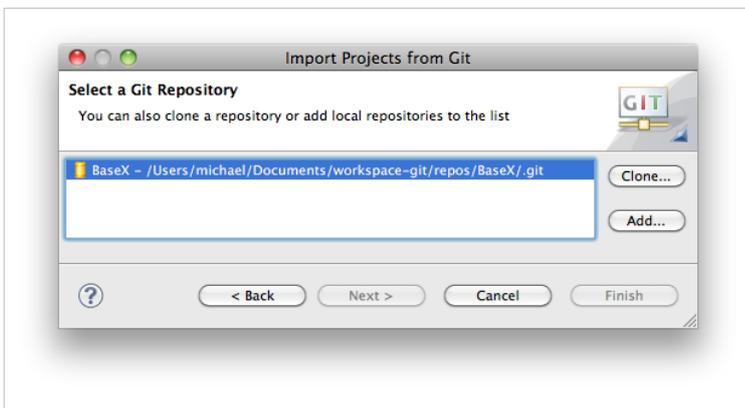
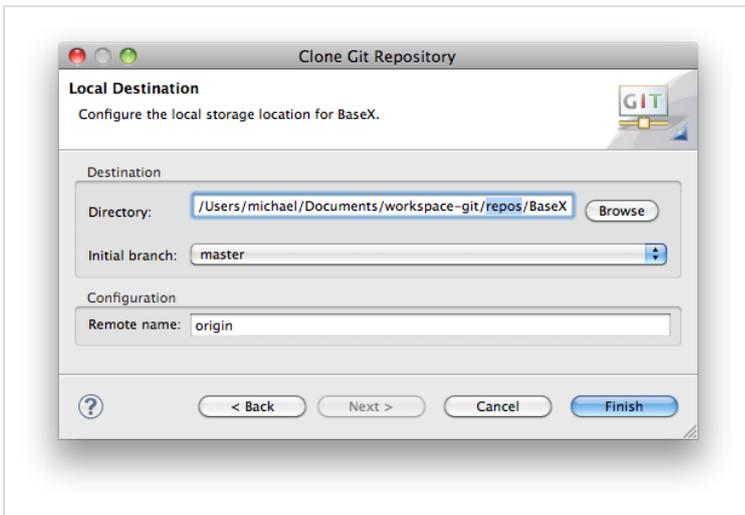
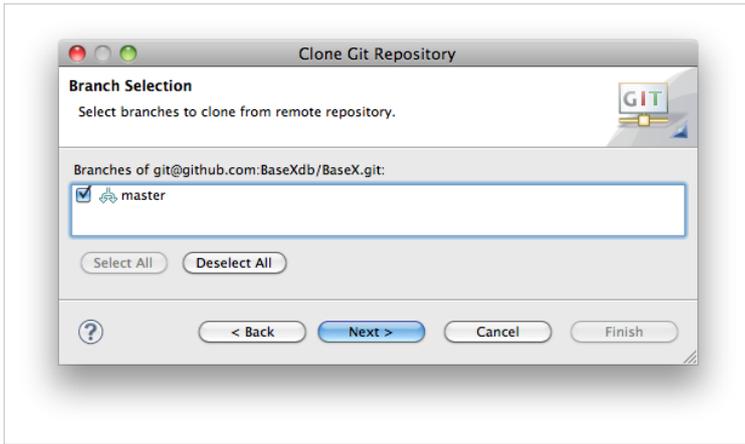
Using Git on Command-Line

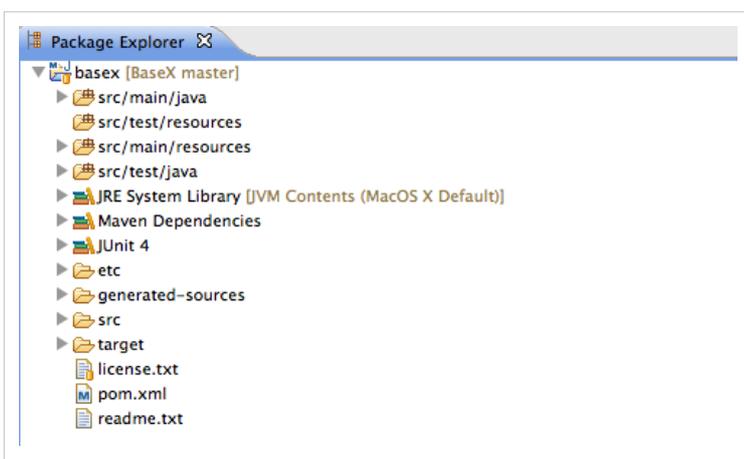
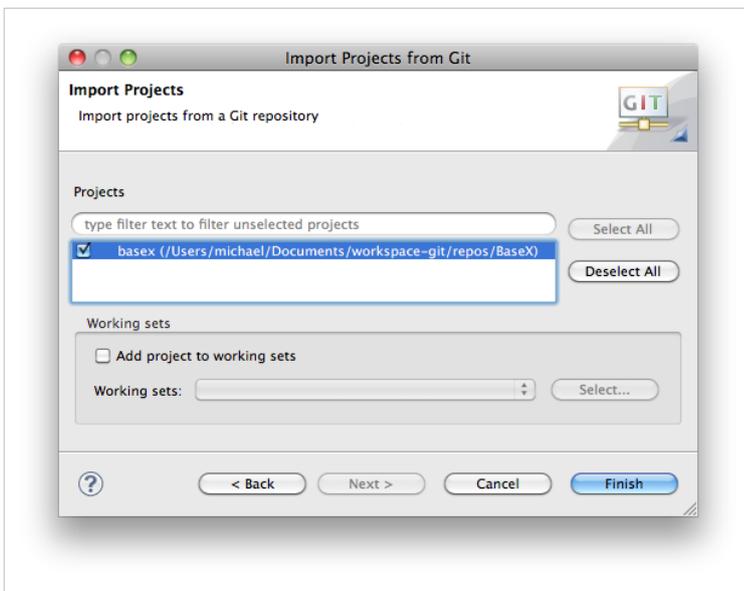
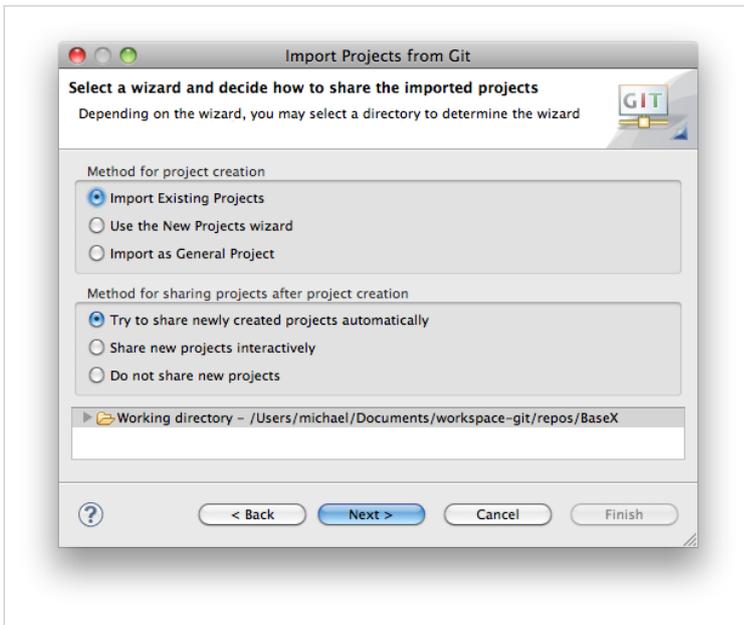
Note: this is not intended to be a complete git reference; it's purpose is to quickly introduce BaseX developers to the most commonly used git commands in the context of the BaseX project.

Preparation

1. Create a GitHub user account: here ^[6] (your github user name will be referenced as \$username)
2. Set up SSH access to GitHub as described here ^[7]
3. Create a fork of one of the BaseXdb projects (it will be referenced as \$project)
4. Choose a directory where the project will be created and make it your working

directory (e. g. /home/user/myprojects)





Clone Your Personal Repository

```
$ git clone git@github.com:$username/$project.git
Cloning into $project...
Enter passphrase for key '/home/user/.ssh/id_rsa':
...

$ ls -d -l $PWD/*
/home/user/myprojects/$project
```

Note that git automatically creates a directory where the repository content will be checked out.

List Remote Repositories

```
$ git remote -v
origin  git@github.com:$username/$project.git (fetch)
origin  git@github.com:$username/$project.git (push)
```

Currently, there is only one remote repository; it is automatically registered during the clone operation. Git remembers this repository as the default repository for push/pull operations.

List Local Changes

After some files have been changed locally, the changes can be seen as follows:

```
$ git diff
diff --git a/readme.txt b/readme.txt
index fabaeaa..cd09568 100644
--- a/readme.txt
+++ b/readme.txt
@@ -49,6 +49,10 @@ ADDING CHECKSTYLE -----
- Enter the URL: http://eclipse-cs.sourceforge.net/update
- Follow the installation procedure and restart Eclipse

+USING GIT -----

Any kind of feedback is welcome; please check out the online documentation at
```

Commit to Local Repository

Note: this commit operation does **not** commit into the remote repository!

First, it is needed to select the modified files which should be committed:

```
$ git add readme.txt
```

Then perform the actual commit:

```
$ git commit
[master 0fdelfb] Added TODO in section "USING GIT"
1 files changed, 4 insertions(+), 0 deletions(-)
```

Before executing the actual commit, git will open the default shell editor (determined using the \$EDITOR variable, usually vi) to enter a message describing the commit changes.

Alternative way is to commit all changed files, i. e. it is not needed to explicitly add the changed files:

```
$ git commit -a
[master 0fdelfb] Added TODO in section "USING GIT"
 1 files changed, 4 insertions(+), 0 deletions(-)
```

Pushing Local Changes to Remote Repository

```
$ git push
Enter passphrase for key '/home/user/.ssh/id_rsa':
Everything up-to-date
```

Pulling Changes from Remote Repository

```
$ git pull
Enter passphrase for key '/home/user/.ssh/id_rsa':
Already up-to-date.
```

Add BaseXdb Upstream Repository

The upstream repository is the one from which the BaseX releases are made and the one from which the personal repository was forked.

```
$ git remote add upstream git@github.com:BaseXdb/$project.git

$ git remote -v
origin  git@github.com:$username/$project.git (fetch)
origin  git@github.com:$username/$project.git (push)
upstream      git@github.com:BaseXdb/$project.git (fetch)
upstream      git@github.com:BaseXdb/$project.git (push)
```

Pulling Changes from Upstream to Local Repository

When some changes are made in the upstream repository, they can be pulled to the local repository as follows:

```
$ git pull upstream master
Enter passphrase for key '/home/user/.ssh/id_rsa':
From github.com:BaseXdb/$project
 * branch          master      -> FETCH_HEAD
Already up-to-date.
```

The changes can then be pushed in the personal repository:

```
$ git push
```

Check out the links at the end of the page for more git options.

Need help using git?

Installing

For information on how to install git on various platforms please refer to: [GitHub: git Installation Guide](#) ^[8]

Documentation

- [Comprehensive Getting Starting Guide on GitHub](#) ^[9]
- [The git book](#) ^[10]
- [Gitcasts.com – Video Guides](#) ^[11]

References

- [1] <http://git-scm.com/>
- [2] <https://github.com>
- [3] <http://www.jcraft.com/jsch>
- [4] https://bugs.eclipse.org/bugs/show_bug.cgi?id=326526
- [5] <http://egit.eclipse.org/r/#change,2037>
- [6] <https://github.com/signup/free>
- [7] <http://help.github.com/key-setup-redirect>
- [8] <http://help.github.com/git-installation-redirect/>
- [9] <http://help.github.com/>
- [10] <http://book.git-scm.com/index.html>
- [11] <http://gitcasts.com/>

Maven

This page is part of the Developer Section. It demonstrates how Maven ^[1] is used to compile and run BaseX, and embed it into other projects.

Using Maven

If you have cloned our repositories and installed Maven on your machine, you can run the following commands from all local repository directories:

- `mvn compile`: the BaseX source files are compiled. To launch the project, type in `java -cp target/classes org.basex.BaseX`, or have a look at our Start Scripts.
- `mvn package`: JAR archives are created in the `target class` directory.
- `mvn install`: the JAR archive is installed to the local repository, and made available to other Maven projects. This is particularly useful if you are compiling a beta version of BaseX, for which no archives exist in the repositories.

By adding the flag `-DskipTests=true` you can skip running the JUnit tests and speed up the packaging procedure. You may as well use Eclipse and m2eclipse to compile the BaseX sources.

Artifacts

You can easily embed BaseX into your own Maven projects by adding the following XML snippets to your pom.xml file:

```
<repositories>
  <repository>
    <id>basex</id>
    <name>BaseX Maven Repository</name>
    <url>http://files.basex.org/maven</url>
  </repository>
</repositories>
```

BaseX Main Package

```
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex</artifactId>
  <version>7.2.1</version>
</dependency>
```

BaseX APIs

...including APIs and the REST, RESTXQ and WebDAV services:

```
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
  <version>7.2.1</version>
</dependency>
```

References

[1] <http://maven.apache.org>

Releases

This page is part of the Developer Section. It lists the official locations of major and minor BaseX versions:

Official Releases

Our releases, packaged for various platforms, are linked from our homepage. They are updated every 2-8 weeks:

- <http://basex.org/download>

Our file server contains links to older releases as well (but we recommend everyone to stay up-to-date, as you'll get faster feedback working with the latest version):

- <http://files.basex.org/releases>

Stable Snapshots

If you are a developer, we recommend you to regularly download one of our stable snapshots, which are packaged and uploaded several times a week:

- <http://files.basex.org/releases/latest/>

Note that the offered snapshot files are replaced as soon as newer versions are available.

Code Base

If you always want to be on the cutting edge, you are invited to watch and clone our GitHub repository:

- <https://github.com/BaseXdb/>

We do our best to keep our main repository stable as well.

Maven Artifacts

The official releases and the current snapshots of both our core and our API packages are also deployed as Maven artifacts on our file server at regular intervals:

- <http://files.basex.org/maven/org/basex/>

Linux

BaseX can also be found in some Linux distributions, such as Debian, Ubuntu and archlinux (Suse and other distributions will follow soon):

- Debian: <http://packages.debian.org/sid/basex>
 - Ubuntu: <http://launchpad.net/ubuntu/+source/basex>
 - Arch Linux: <http://aur.archlinux.org/packages.php?ID=38645>
-

Translations

This page is part of the Developer Section. It describes how to translate BaseX into other (natural) languages.

BaseX is currently available in **nine** languages, namely English, German, Japanese, French, Italian, Dutch, Indonesian, Vietnamese and Mongolian.

It is easy to translate BaseX into your native language! This is how you can proceed:

Using the JAR file

JAR files are nothing else than ZIP archives. All language files are placed in the `lang` directory inside the JAR file:

1. Unzip the source file `English.lang` and rename it to your target language (e.g. `Zulu.lang`)
2. Enter your name and potential contact information in the second line and translate all texts
3. Update your JAR file by copying the translated file into the zipped `lang` directory. Your new language file will be automatically scanned.
4. Start BaseX and choose your language in the GUI (*Options* → *Preferences...*)
5. Restart BaseX to see the changes

Using the sources

All language files are placed in the `src/main/resources/lang` directory of the main project:

1. Create a copy of the source file `English.lang` and rename it to your target language (e.g. `Zulu.lang`)
2. Enter your name and potential contact information in the second line and translate all texts
3. Copy the translated file back to the `lang` directory. Your new language file will be automatically scanned.
4. Recompile the project, start BaseX and choose your language in the GUI (*Options* → *Preferences...*)
5. Restart BaseX to see the changes

You may as well change the language in the `.basex` configuration file, which is placed in your home directory. To see where the all text keys are used within BaseX, we recommend you to temporarily set the `LANGKEY` option to `true`.

HTTP Services

REST

This page is part of the Developer Section. It describes how to use the REST API of BaseX.

BaseX offers a RESTful API for accessing distributed XML resources. REST (REpresentational State Transfer ^[1]) facilitates a simple and fast access to databases through HTTP. The HTTP methods GET, PUT, DELETE, and POST can be used to interact with the database.

The REST implementation has formerly been based on JAX-RX, a generic layer to provide unified access to XML databases and resources. With [Version 7.0](#), it has been replaced with a native REST implementation that allows a closer integration with XQuery, WebDAV, and other features of BaseX. If you have worked with JAX-RX before, please take some time to understand the differences between the old and new API.

Getting Started

First of all, launch the BaseX as Web Application. By default, Jetty ^[3] is used as web server. All HTTP services will be available on port 8984, and the REST service is accessible at `http://localhost:8984/rest/`. If the server is started as servlet, all Main Options (such as the path to the database) can be configured in the `web.xml` file. If run as a standalone application, the settings are stored in the file `.basex`.

Most browsers can be used to get to know the REST API, and to directly display the results. When requesting the first result, you will have to enter some database credentials (default: `admin/admin`, see below). Some alternatives for using REST are listed in the Usage Examples.

URL Architecture

The root URL lists all available databases. The following examples assume that you have created a database instance from the `factbook.xml` ^[3] document:

`http://localhost:8984/rest`

```
<rest:databases resources="1" xmlns:rest="http://basex.org/rest">
  <rest:database resources="1" size="1813599">factbook</rest:database>
</rest:databases>
```

The resources of a database can be listed by specifying the database, and potential sub directories, in the URL. In the given example, a single XML document is stored in the *factbook* database:

`http://localhost:8984/rest/factbook`

```
<rest:database name="factbook" resources="1" xmlns:rest="http://basex.org/rest">
  <rest:resource type="xml" content-type="application/xml" size="77192">factbook.xml</rest:resource>
</rest:database>
```

The contents of a database can be retrieved by directly addressing the resource:

`http://localhost:8984/rest/factbook/factbook.xml`

If a resource is not found, an HTTP response will be generated with 404 as status code.

Operations

GET and POST requests support the following **operations**:

- **Query:**
Evaluates an XPath/XQuery expression. If a database or database path is specified in the URL, it is used as initial query context.
- **Run:**
Runs a query file located on the server. The query directory is defined by the `HTTPPATH` option.
- **Command:**
Executes a database command.
- **Get:**
This is the default operation for the GET operation (it is not possible to use this operation in a POST request). It returns a list of all databases, the resources of a database or the addressed resource.

Parameters

Additionally, the following **parameters** can be applied to the operations:

- **Variables:**
External variables can be *bound* before a query is evaluated (see below for more).
- **Context:**
The `context` parameter may be used to provide an initial XML context node.
- **Options:**
Specified Options are applied before the actual operation will be performed.
- **Serialization:**
All Serialization parameters known to BaseX can be specified as query parameters. Parameters that are specified within a query will be interpreted by the REST server before the output is generated.
- **Wrap:**
The `wrap` parameter encloses all query results with XML elements, using the `http://basex.org/rest` namespace.

While **Options** can be specified for all operations, the remaining parameters will only make sense for **Query** and **Run**.

Request Methods

GET Requests

If the GET method is used, all query parameters are directly specified within the URL.

Examples

- The first example lists all resources found in the **tmp** path of the *factbook* database:
`http://localhost:8984/rest/factbook/tmp`
- The first example prints the city names from the *factbook* database and encloses all results with a `<rest:result/>` elements:
`http://localhost:8984/rest/factbook?query=//city/name&wrap=yes`
- In the next request, `US-ASCII` is chosen as output encoding, and the query `eval.xq` is evaluated:
`http://localhost:8984/rest?run=eval.xq&encoding=US-ASCII`
- The next URL turns on XML wrapping and lists all database users that are known to BaseX:
`http://localhost:8984/rest?command=show+users`

- The last example includes an to disallow XQuery 3.0 expressions:
http://localhost:8984/rest?query=12345&xquery3=false

POST Requests

The body of a POST request is interpreted as XML fragment, which specifies the operation to perform. The body must conform to a given XML Schema.

Examples

- The following query returns the first five city names of the **factbook** database:

```
<query xmlns="http://basex.org/rest">
  <text><![CDATA[ (//city/name) [position() <= 5] ]]></text>
</query>
```

- The second query returns the string lengths of all text nodes, which are found in the node that has been specified as initial context node:

```
<rest:query xmlns:rest="http://basex.org/rest">
  <rest:text>for $i in ./text() return string-length($i)</rest:text>
  <rest:context>
    <xml>
      <text>Hello</text>
      <text>World</text>
    </xml>
  </rest:context>
</rest:query>
```

- The following request returns the registered database users encoded in ISO-8859-1:

```
<command xmlns="http://basex.org/rest">
  <text>show users</text>
  <parameter name='encoding' value='ISO-8859-1' />
</command>
```

- This example creates a new database from the specified input and retains all whitespaces:

```
<command xmlns="http://basex.org/rest">
  <text>create db test http://files.basex.org/xml/xmark.xml</text>
  <option name='chop' value='false' />
</command>
```

- The last request runs a query `query.xq` located in the directory specified by HTTPPATH:

```
<run xmlns="http://basex.org/rest">
  <text>query.xq</text>
</run>
```

PUT Requests

The PUT method is used to create new databases, or to add or update existing database resources:

- **Create Database:**
A new database is created if the URL only specifies the name of a database. If the request body contains XML, a single document is created, adopting the name of the database.
- **Store Resource:**
A resource is added to the database if the URL contains a database path. If the addressed resource already exists, it is replaced by the new input.

There are two ways to store non-XML data in BaseX:

- **Store as raw:**
If application/octet-stream is chosen as content-type the input data is added as raw.
- **Convert to XML:**
Raw data can be explicitly converted to XML by specifying the content-type.

Trying to add raw data without specifying the content type or specifying a wrong content type will eventually lead to a 400 (BAD REQUEST) exception. The following content types are available:

- **application/octet-stream:** Stores input data as raw file.
- **application/json:** Stores JSON as XML.
- **application/jsonml:** Stores JSONML input as XML.
- **text/plain:** Stores plain text input as XML.
- **text/comma-separated-values:** Stores CSV text input as XML.
- **text/html:** Stores HTML input as XML.

Examples

- A new database with the name **XMark** is created. If XML input is sent in the HTTP body, the resulting database resource will be called **XMark.xml**:
`http://admin:admin@localhost:8984/rest/XMark`
- A new database is created, and no whitespaces will be removed from the passed on XML input:
`http://admin:admin@localhost:8984/rest/XMark?chop=false`
- The contents of the HTTP body will be taken as input for the document **one.xml**, which will be stored in the **XMark** database:
`http://admin:admin@localhost:8984/rest/XMark/one.xml`

An HTTP response with status code 201 (CREATED) is sent back if the operation was successful. Otherwise, the server will reply with 404 (if a specified database was not found) or 400 (if the operation could not be completed).

Have a look at the usage examples for more detailed examples using Java and shell tools like cURL.

DELETE Requests

The DELETE method is used to delete databases or resources within a database.

Example

- The **factbook** database is deleted:
`http://admin:admin@localhost:8984/rest/factbook`
- All resources of the **XMark** database are deleted that reside in the **tmp** path:
`http://admin:admin@localhost:8984/rest/XMark/tmp/`

The HTTP status code 404 is returned if no database is specified. 200 (OK) will be sent in all other cases.

Assigning Variables

GET Requests

All query parameters that have not been processed before will be treated as variable assignments:

Examples

- The following request binds a single variable to the query to be processed:
`http://localhost:8984/rest?query=$text&text=Hello+World`
- The following request assigns two variables to a server-side query file `mult.xq` placed in the database DB:
`http://localhost:8984/rest/DB/mult.xq?a=21&b=2`

```
(: XQuery file: mult.xq :)
declare variable $a as xs:integer external;
declare variable $b as xs:integer external;
$a * $b
```

Variables can also be explicitly prefixed with a dollar sign (\$); this way, those variables can be bound as well that would otherwise be interpreted in a different way (e.g.: `$method`).

POST Requests

If `query` or `run` is used as operation, external variables can be specified via the `<variable/>` element:

```
<query xmlns="http://basex.org/rest">
  <text>
    declare variable $x as xs:integer external;
    declare variable $y as xs:integer external;
    $x * $y
  </text>
  <variable name="x" value="21"/>
  <variable name="y" value="2"/>
</query>
```

User Management

By default, the HTTP server is started with no predefined user. Users and passwords can be sent via HTTP basic authentication ^[2] with each HTTP request. As alternative, users and passwords can also be stored server-side in the "org.basex.user" and "org.basex.password" system properties before the HTTP server is started, or specified as command-line arguments.

With cURL, and most browsers, you can specify the user name and password with each HTTP request within the request string as plain text, using the format `USER:PASSWORD@URL`. An example:

```
http://admin:admin@localhost:8984/rest/factbook
```

Content Type

As the content type of a REST response cannot be dynamically determined in all cases, it can be manually adjusted by the user. The final content type of a REST response is chosen in several steps:

1. By default, the content type of a response depends on the chosen operation:
 - **Query/Run** ? application/xml
 - **Command** ? text/plain
 - **Get** ? application/xml, or content type of the addressed resource
2. The default content type is overwritten if a serialization method is specified, either as query parameter or within the XQuery expression. The following methods are available:
 - xml ? application/xml
 - xhtml ? text/html
 - html ? text/html
 - text ? text/plain
 - raw ? application/octet-stream
 - json or jsonml ? application/json
3. The content type is overwritten in any case if a specific media-type is chosen, again as query parameter or within the query.

The following three example requests will all return `<a/>` as result and use `application/xml` as content-type:

```
http://localhost:8984/rest?query=%3Ca/%3E
http://localhost:8984/rest?query=%3Ca/%3E&method=xml
http://localhost:8984/rest?query=%3Ca/%3E&media-type=application/xml
```

Usage Examples

Java

Authentication

Most programming languages offer libraries to communicate with HTTP servers. The following example demonstrates how easy it is to perform a DELETE request with Java.

Basic access authentication can be activated in Java by adding an authorization header to the `HttpURLConnection` instance. The header contains the word `Basic`, which specifies the authentication method, followed by the Base64-encoded `USER:PASSWORD` pair. As Java does not include a default conversion library for Base64 data, the internal BaseX class `org.basex.util.Base64` can be used for that purpose:

```
// The java URL connection to the resource.
URL url = new URL("http://admin:admin@localhost:8984/rest/factbook");

// Establish the connection to the URL.
URLConnection conn = (URLConnection) url.openConnection();
// Set as DELETE request.
conn.setRequestMethod("DELETE");

// User and password.
String user = "bob";
String pw = "alice";
// Encode user name and password pair with a base64 implementation.
String encoded = Base64.encode(user + ":" + pw);
// Basic access authentication header to connection request.
conn.setRequestProperty("Authorization", "Basic " + encoded);

// Print the HTTP response code.
System.out.println("HTTP response: " + conn.getResponseCode());

// Close connection.
conn.disconnect();
```

Content-Types

The content-type of the input can easily be included, just add the following property to the connection (in this example we explicitly store the input file as raw):

```
// store input as raw
conn.setRequestProperty("Content-Type", "application/octet-stream");
```

See the PUT Requests section for a description of the possible content-types.

Find Java examples for all methods here: GET ^[3], POST ^[4], PUT ^[5], DELETE ^[6].

Command Line

Tools such as the Linux commands Wget ^[7] or cURL ^[8] exist to perform HTTP requests (try copy & paste):

GET

- `curl -i "admin:admin@localhost:8984/rest/factbook?query=//city/name"`

POST

- `curl -i -X POST -H "Content-Type: application/xml" -d "<query xmlns='http://basex.org/rest'><text>//city/name</text></query>" "admin:admin@localhost:8984/rest/factbook"`
- `curl -i -X POST -H "Content-Type: application/xml" -T query.xml "admin:admin@localhost:8984/rest/factbook"`

PUT

- `curl -i -X PUT -T "etc/xml/factbook.xml" "admin:admin@localhost:8984/rest/factbook"`

- `curl -i -X PUT -H "Content-Type: text/plain" -T "etc/xml/factbook.xml" "admin:admin@localhost:8984/rest/factbook"`

DELETE

- `curl -i -X DELETE "admin:admin@localhost:8984/rest/factbook"`

Changelog

Version 7.2

- Removed: direct evaluation of addresses resources with `application/xquery` as content type

Version 7.1.1

- Added: `options` parameter for specifying database options

Version 7.1

- Added: PUT request: automatic conversion to XML if known content type is specified

Version 7.0

- REST API introduced, replacing the old JAX-RX API

References

- [1] http://en.wikipedia.org/wiki/Representational_State_Transfer
 - [2] http://en.wikipedia.org/wiki/Basic_access_authentication
 - [3] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples/rest/RESTGet.java>
 - [4] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples/rest/RESTPost.java>
 - [5] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples/rest/RESTPut.java>
 - [6] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples/rest/RESTDelete.java>
 - [7] <http://www.gnu.org/s/wget/>
 - [8] <http://curl.haxx.se/>
-

REST: POST Schema

The following schema is used from the REST API to validate POST requests:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://basex.org/rest"
  targetNamespace="http://basex.org/rest">
  <xs:element name="query">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="variable" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="context" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="run">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="variable" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="context" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="command">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="text" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="option" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="text" type="xs:string"/>
  <xs:element name="option">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="parameter">
```

```

<xs:complexType>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="variable">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="value" type="xs:string" use="required"/>
    <xs:attribute name="type" type="xs:string" use="optional"/>
  </xs:complexType>
</xs:element>
<xs:element name="context" type="xs:anyType"/>
</xs:schema>

```

RESTXQ

This page is part of the Developer Section. It describes how to use the RESTXQ API of BaseX.

RESTXQ, introduced by Adam Retter ^[1], is a new API that facilitates the use of XQuery as a Server Side processing language for the Web. RESTXQ has been inspired by Java's JAX-RS API ^[2]: it defines a pre-defined set of XQuery 3.0 annotations for mapping HTTP requests to XQuery functions, which in turn generate and return HTTP responses.

As of [Version 7.2](#), RESTXQ is supported by BaseX. Note that various details of the specification may be subject to change due to the early state of the API.

Getting started

First of all, launch the BaseX as Web Application. By default, Jetty ^[3] is used as web server. All HTTP services will be available on port 8984, and the RESTXQ service is accessible at `http://localhost:8984/restxq/`. If the server is started as servlet, all Main Options (such as the path to the database) can be configured in the `web.xml` file. If run as a standalone application, the settings are stored in the file `.basex`.

Module Declarations

A RESTXQ module needs to contain a declaration to the namespace `http://exquery.org/ns/restxq`. A *Resource Function* is an XQuery function that has been marked up with RESTXQ annotations (annotations will be introduced with the upcoming XQuery 3.0 standard). When an HTTP request comes in, a resource function will be invoked that matches the constraints indicated by its annotations.

All files with extension `*.xqm`, placed inside the directory specified by `HTTPPATH` (refer to HTTP Server Configuration) will be treated as RESTXQ modules. These will be parsed for RESTXQ annotations and cached. If a module is modified, it will be parsed again at runtime.

A simple RESTXQ module is shown below, it is part of a clean installation and available at `http://localhost:8984/restxq/`.

```

(:~ simplified module as in http/restxq.xqm :)
module namespace page = 'http://basex.org/modules/web-page';
declare namespace rest = 'http://exquery.org/ns/restxq';

```

```

declare %rest:path("hello/{$world}")
  %rest:GET
  %rest:header-param("User-Agent", "{$agent}")
  function page:hello($world as xs:string, $agent as xs:string*) {
<response>
  <title>Hello { $world }!</title>
  <info>You requested this page with { $agent }.</info>
</response>
};

```

If the URI `http://localhost:8984/restxq/hello/world` is accessed, the result will be kind of

```

<response>
  <title>Hello world!</title>
  <info>You requested this page with Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US) AppleWebKit/525.13 (KHTML, like Gecko)
Chrome/0.2.149.27 Safari/525.13.</info>
</response>

```

We added another method within that module:

```

declare %rest:path("form/")
  %rest:POST
  %rest:form-param("content", "{$message}", "'no message delivered'")
  function page:hello-postman($message as xs:string) {
<response>
  <title>Hello!</title>
  <info>It seems you posted a message: { $message }</info>
</response>
};

```

If you posted something (e.g. using curl or the embedded form at `http://localhost:8984/restxq/form`)

```
curl -i -X POST --data "content='Here comes the post'" http://admin:admin@localhost:8984/restxq/form
```

You would receive

```

<response>
  <title>Hello!</title>
  <info>It seems you posted a message: 'Here comes the post'</info>
</response>

```

Annotations

This section lists all annotations provided by RESTXQ. `rest` is used as namespace prefix.

Constraints

Constraints restrict the HTTP requests that a resource function may process.

Paths

A resource function must have a single *Path Annotation*. It will be called if a URL matches the path segments and templates specified as argument. *Path templates* contain variables in curly brackets, and map the values of the actual request path to the function arguments.

The following example contains a path annotation with three segments and two templates. One of the function arguments is further specified with a data type, which means that the value for `$variable` will be cast to an `xs:integer` before being bound:

```
declare %rest:path("/a/path/{$with}/some/{$variable}")
  function($with, $variable as xs:integer) { ... };
```

HTTP Methods

The HTTP method annotations relate to some of the HTTP request methods ^[3] (GET, HEAD, DELETE, POST, PUT). Depending on the request method of the request, a function is invoked.

Simple Method Annotations

All available simple method annotations:

```
%rest:GET
%rest:HEAD
%rest:DELETE
```

Content Method Annotations

The variable declaration, for processing the content in a XQuery function, is optional. All available content method annotations:

```
%rest:POST
%rest:POST ("{$post-body}")
%rest:PUT
%rest:PUT ("{$put-body}")
```

Content Types

- **HTTP Content Types:** One or more media-types may be specified as strings, e.g.:

```
%rest:consumes ("application/xml", "text/xml")
```

- **HTTP Accept:** One or more media-types may be specified as strings, e.g.:

```
%rest:produces ("application/atom+xml")
```

These default to `*/*` if no media-type annotations are given.

Parameters

Parameters are optional annotations that can be used to bind additional values to function arguments:

Query Strings

The value of the *first parameter*, if found in the Query String ^[4], will be assigned to the variable specified as *second parameter*. Optionally, a *third parameter* may be specified as default:

```
%rest:query-param("parameter", "{$value}", "default")
%rest:query-param("answer", "{$answer}", 42, 43, 44)
%rest:query-param("search", "{$search-param}")
```

HTML Form Fields

Form parameters are specified the same way as query strings. Their values are extracted from GET or POST requests.

```
%rest:form-param("parameter", "{$value}", "default")
```

HTTP Headers

Header parameters are specified the same way as query strings:

```
%rest:header-param("User-Agent", "{$user-agent}")
%rest:header-param("Referer", "{$referer}", "none")
```

Cookies

Cookie parameters are specified the same way as query strings:

```
%rest:cookie-param("username", "{$user}")
%rest:cookie-param("authentication", "{$auth}", "no_auth")
```

References

RESTXQ has been proposed by Adam Retter ^[1]. More information on all specifics can be found in the following two documents:

- RESTful XQuery, Standardised XQuery 3.0 Annotations for REST ^[5]. Paper, XMLPrague, 2012
- RESTXQ ^[6]. Slides, MarkLogic User Group London, 2012

References

- [1] <http://www.adamretter.org.uk/>
- [2] http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services
- [3] http://en.wikipedia.org/wiki/HTTP#Request_methods
- [4] http://en.wikipedia.org/wiki/Query_string
- [5] http://www.adamretter.org.uk/papers/restful-xquery_january-2012.pdf
- [6] http://www.adamretter.org.uk/presentations/restxq_mugl_20120308.pdf

WebDAV

This page is part of the Developer Section. It describes how to use the WebDAV file system interface.

BaseX offers access to the databases and documents using the WebDAV ^[1] protocol. WebDAV provides convenient means to access and edit XML documents by representing BaseX databases and documents in the form of a file system hierarchy.

Getting Started

First of all, launch the BaseX as Web Application. By default, Jetty ^[3] is used as web server. All HTTP services will be available on port 8984, and the WebDAV service is accessible at `http://localhost:8984/webdav/`. If the server is started as servlet, all Main Options (such as the path to the database) can be configured in the `web.xml` file. If run as a standalone application, the settings are stored in the file `.basex`.

Generally, the BaseX WebDAV Server can be accessed using either with a `http://<httphost>:<httpport>/webdav/` or `webdav://<httphost>:<httpport>/webdav/` URL, depending on the used WebDAV client.

Authorization

The WebDAV service uses the database user credentials in order to perform authentication and authorization. If database user and password are explicitly specified when starting the BaseX HTTP Server using the corresponding startup options, WebDAV will not request additional user authentication from the client.

WebDAV Clients

Please check out the following tutorials to get WebDAV running on different operating systems and with oXygen:

- Windows 7
- Windows XP
- Mac OSX 10.4+
- GNOME and Nautilus
- KDE
- oXygen Editor

Changelog

Version 7.0

- WebDAV API introduced

References

[1] <http://en.wikipedia.org/wiki/Webdav>

WebDAV: Windows 7

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows 7.

- Open the Explorer
- Open the "Map network drive..." dialog by right-clicking on "My Computer"
- Click on the link "Connect to a Web site that you can use to store your documents and pictures."

What network folder would you like to map?

Specify the drive letter for the connection and the folder that you want to connect to:

Drive: ▼

Folder: ▼

Example: \\server\share

Reconnect at logon

Connect using different credentials

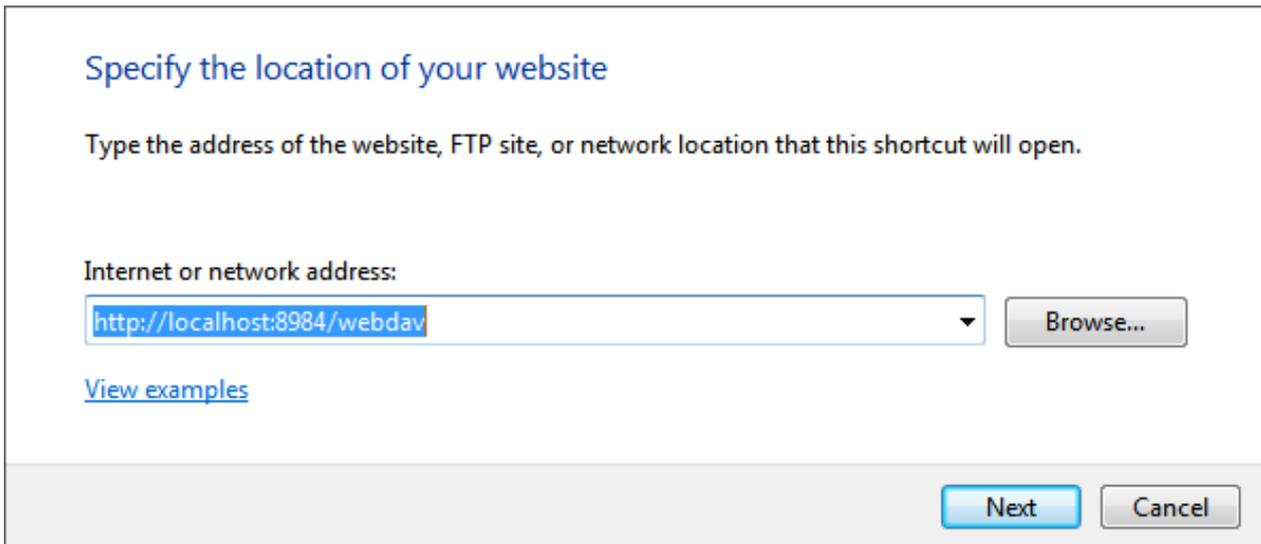
[Connect to a Web site that you can use to store your documents and pictures.](#)

- Click "Next", select "Choose a custom network location" and click "Next" again.

Where do you want to create this network location?

Specify the address of a website, network location, or FTP site.

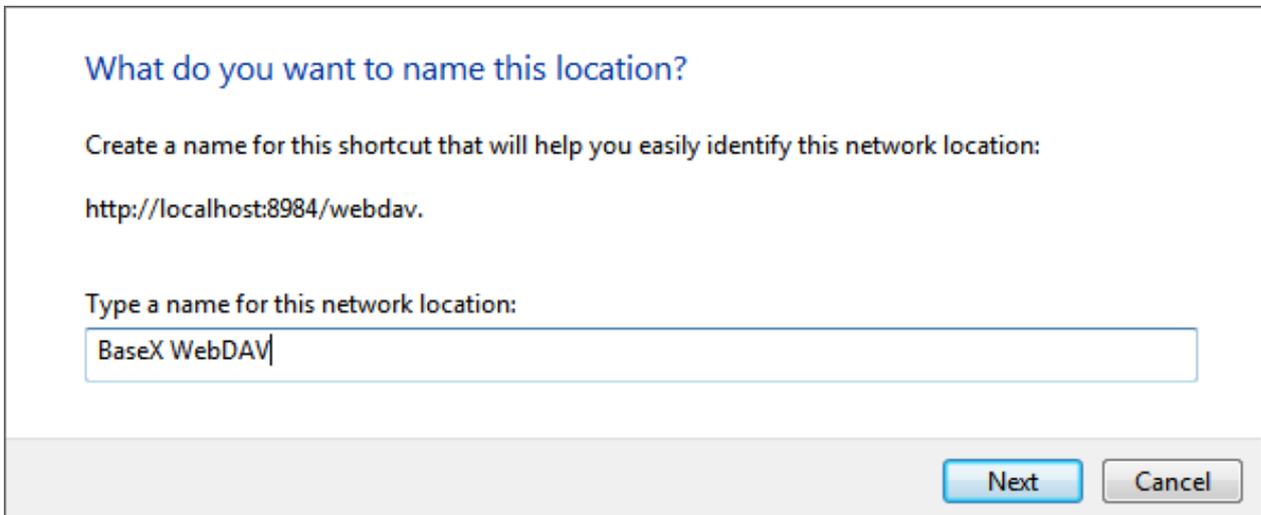
- Enter the URL address of the BaseX WebDAV Server (e.g. <http://localhost:8984/webdav>) and click "Next".



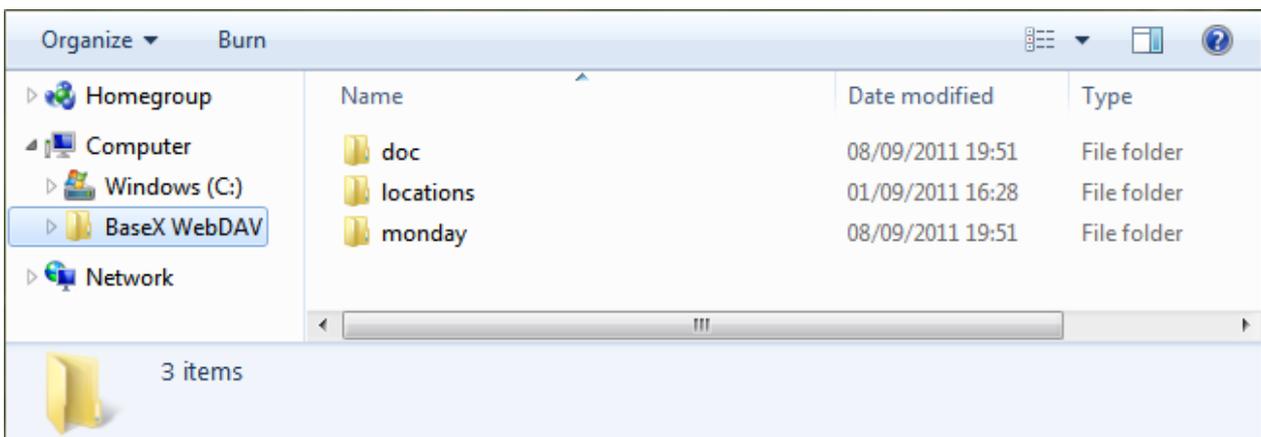
If

a message saying that the folder is not valid, this is because Microsoft WebClient is not configured to use Basic HTTP authentication. Please check this Microsoft article ^[1] in order to enable Basic HTTP authentication.

- Enter a name for the network location and click "Next".



- The BaseX WebDAV can be accessed from the Explorer window.



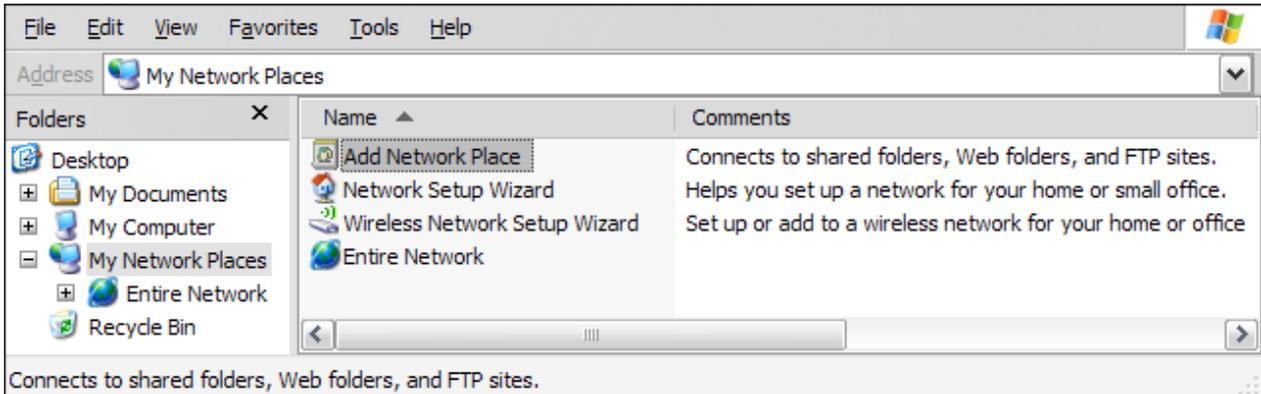
References

[1] <http://support.microsoft.com/kb/928692/en>

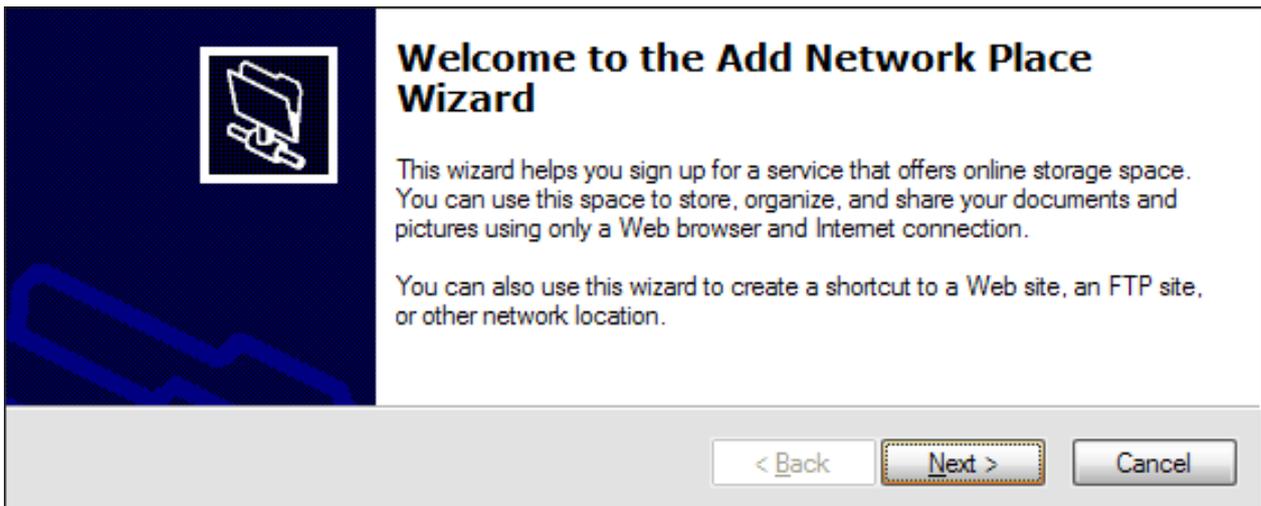
WebDAV: Windows XP

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows XP.

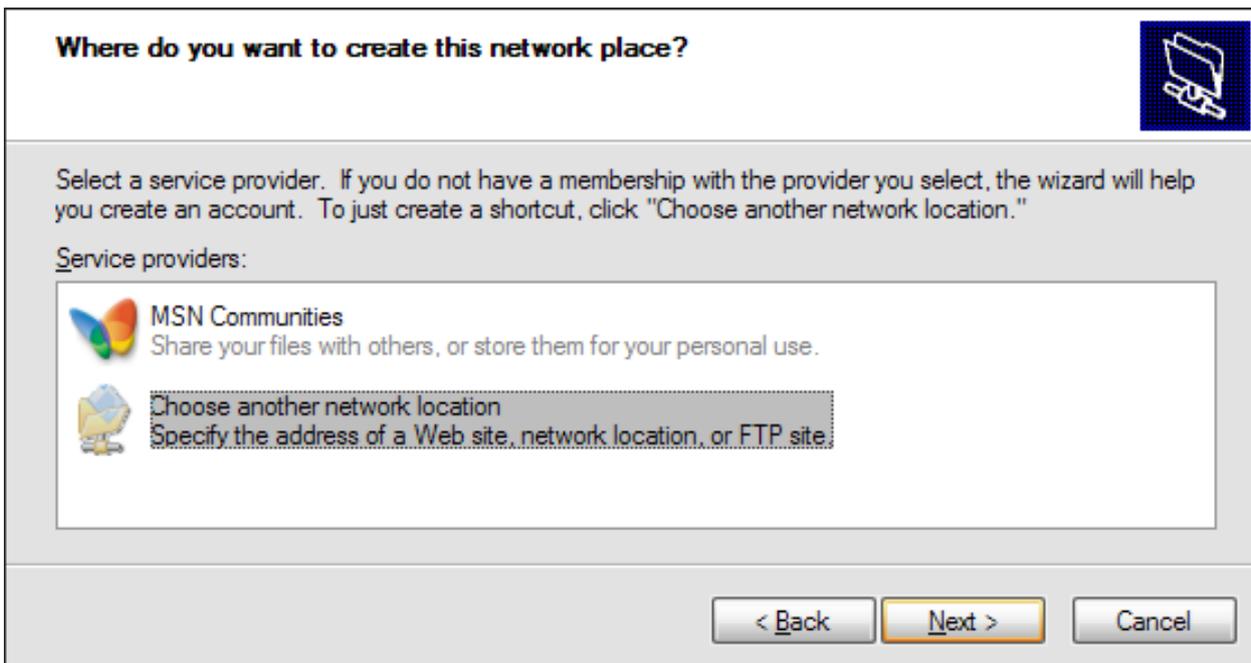
- In the "My Network Places" view, double click on "Add Network Place":



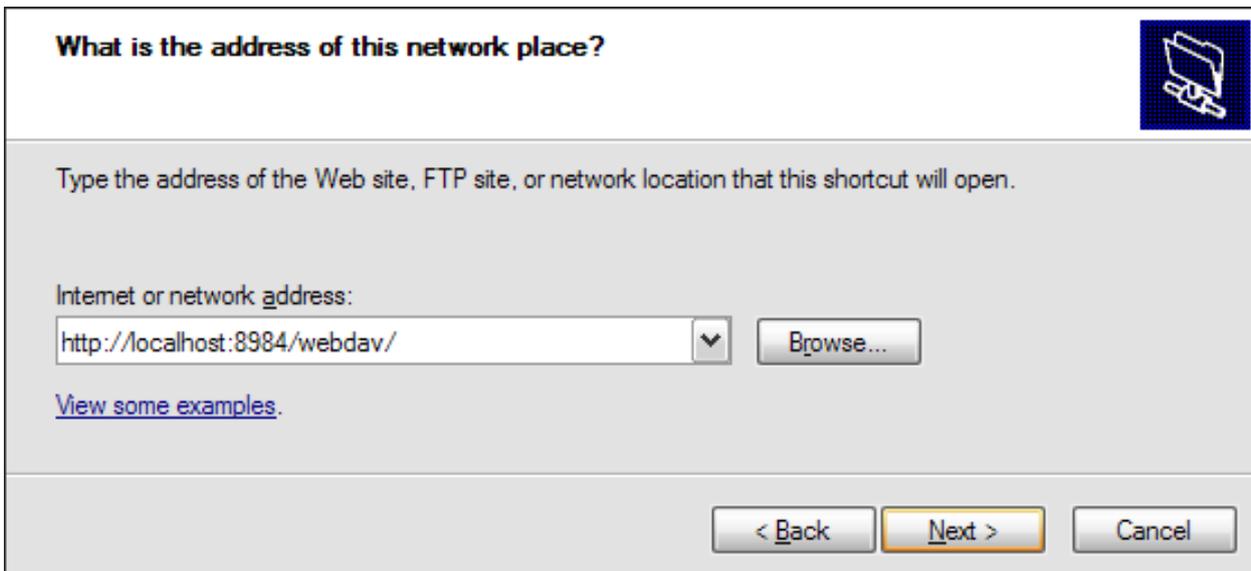
- Confirm the upcoming introductory dialog:



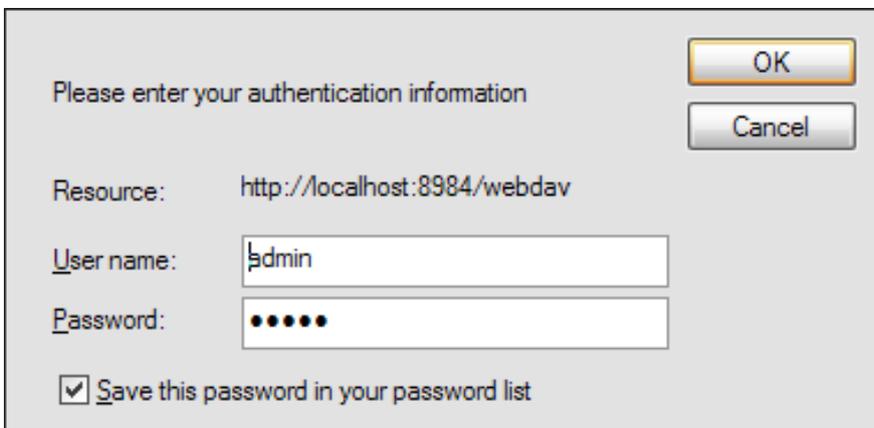
- Select "Choose another network location" in the next dialog:



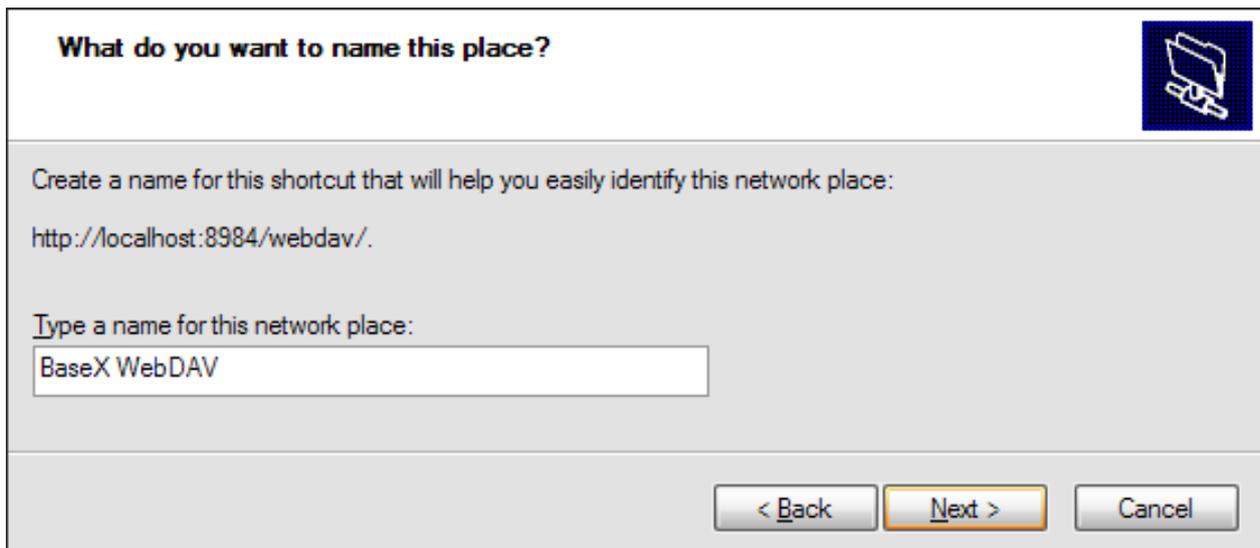
- Next, specify the BaseX WebDAV URL:



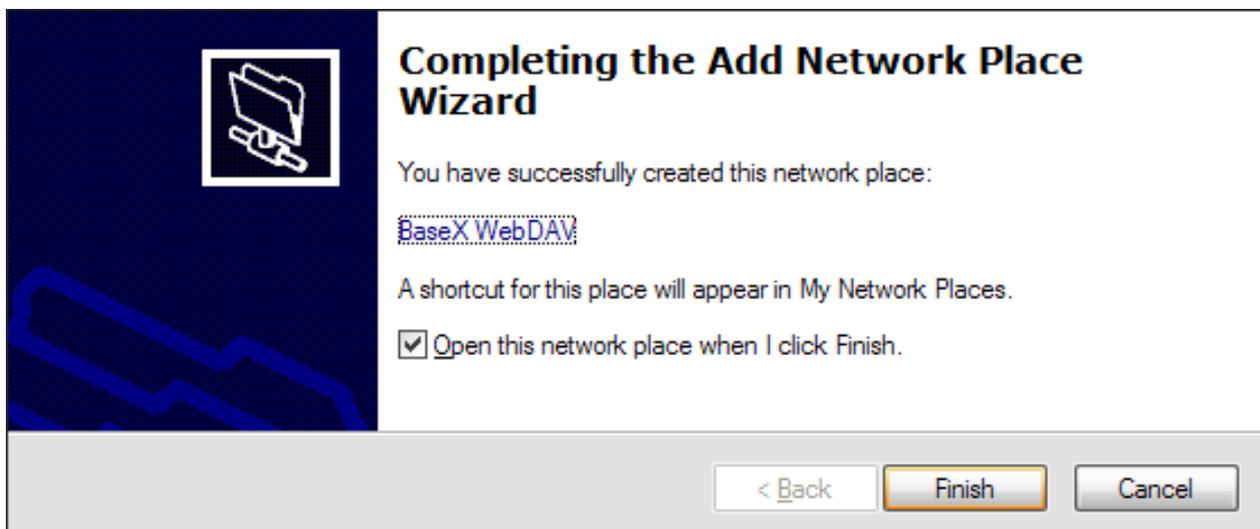
- Enter the user/password combination to connect to the WebDAV service:



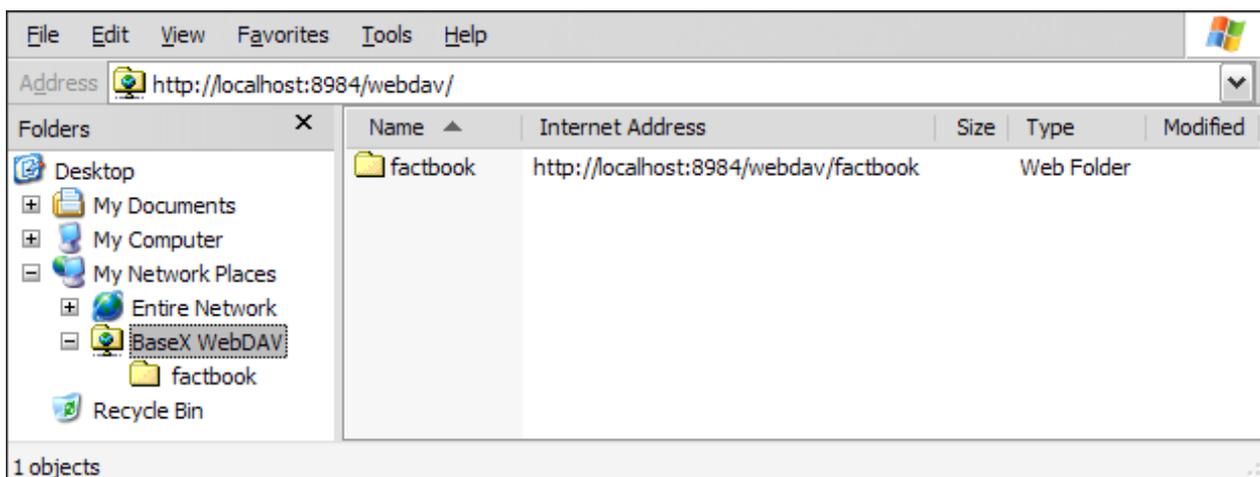
- Assign a name to your WebDAV connection:



- Finish the wizard:



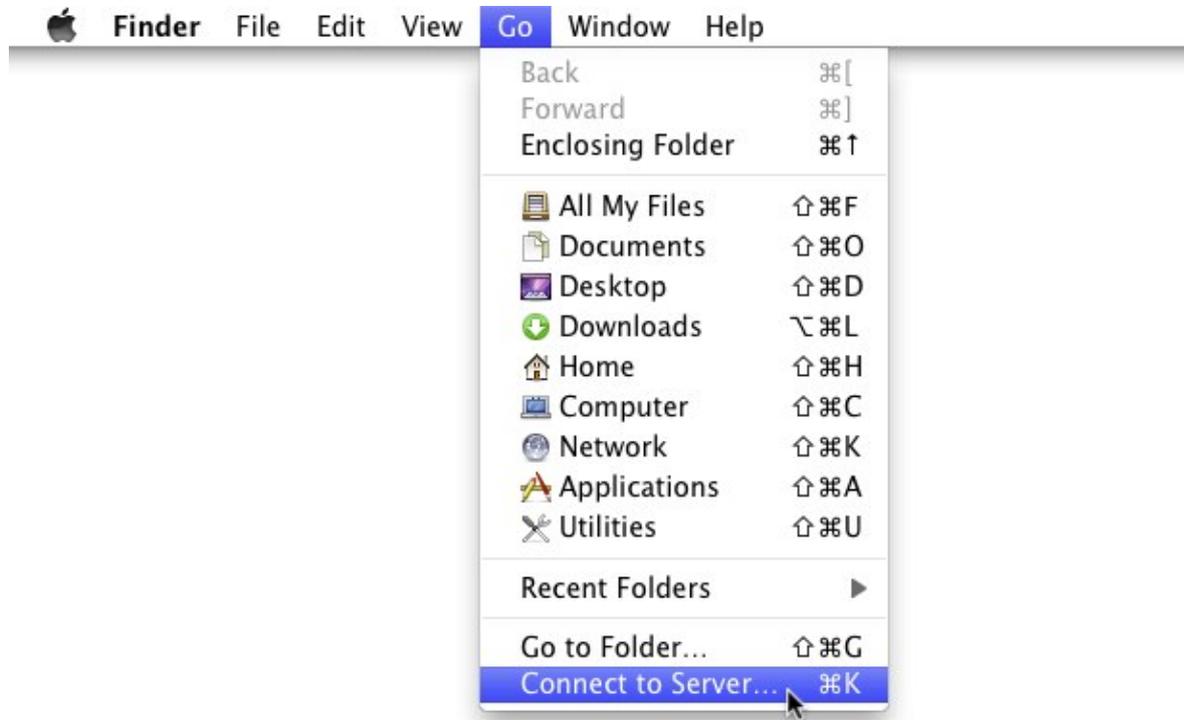
- You can now see all BaseX databases in the Windows Explorer:



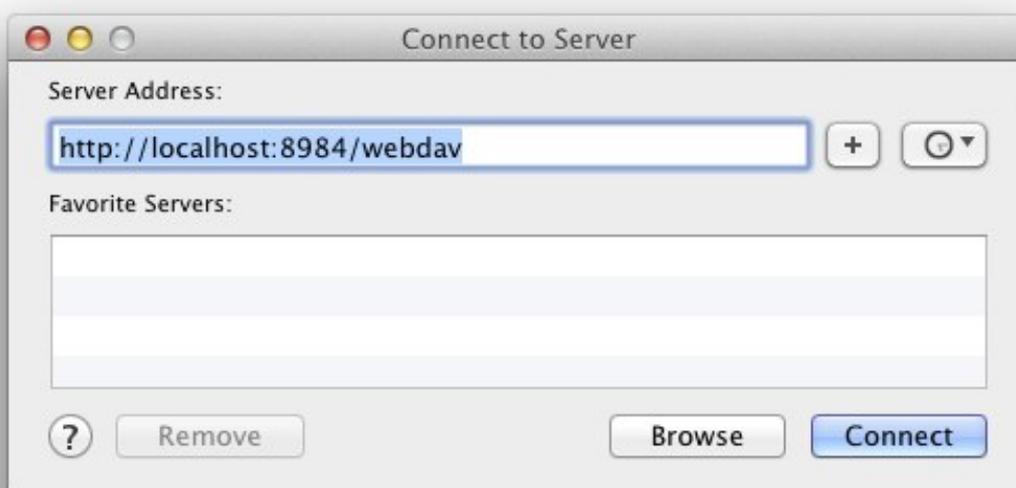
WebDAV: Mac OSX

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Mac OS X 10.4+.

- Mac OS X supports WebDAV since 10.4/Tiger
- Open Finder, choose Go -> Connect to Server:



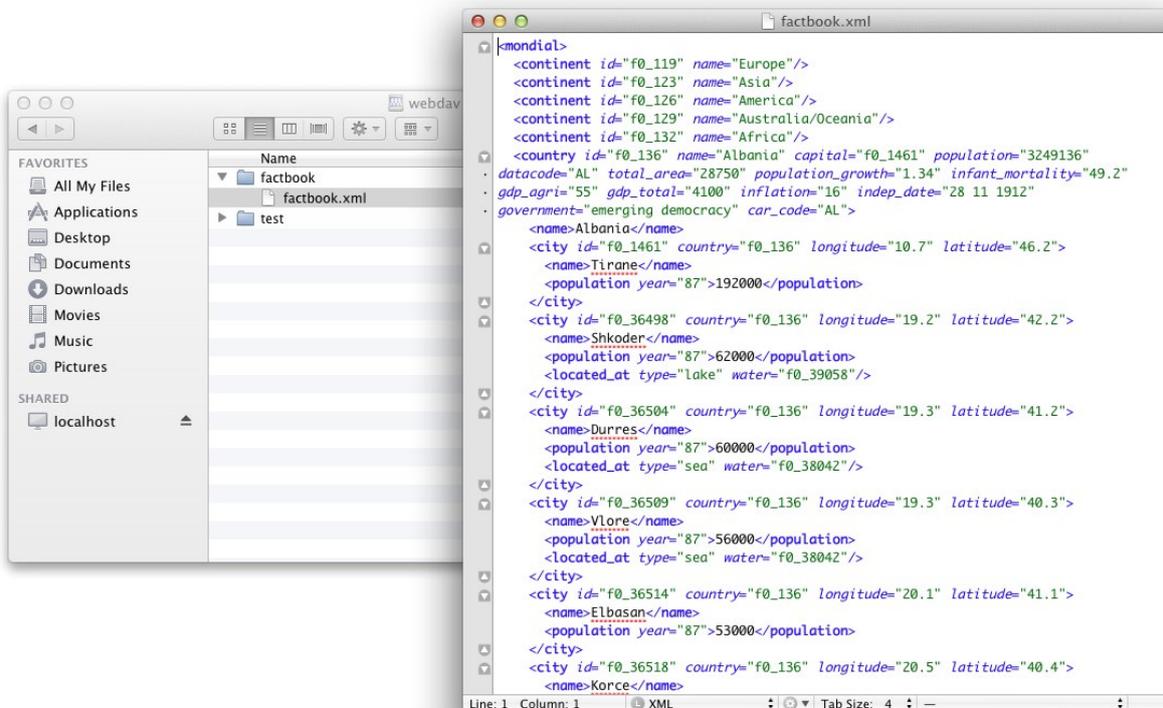
- Enter BaseX WebDAV URL (eg. <http://localhost:8984/webdav>) - do not use webdav://-scheme! Press Connect:



- Enter the user credentials:



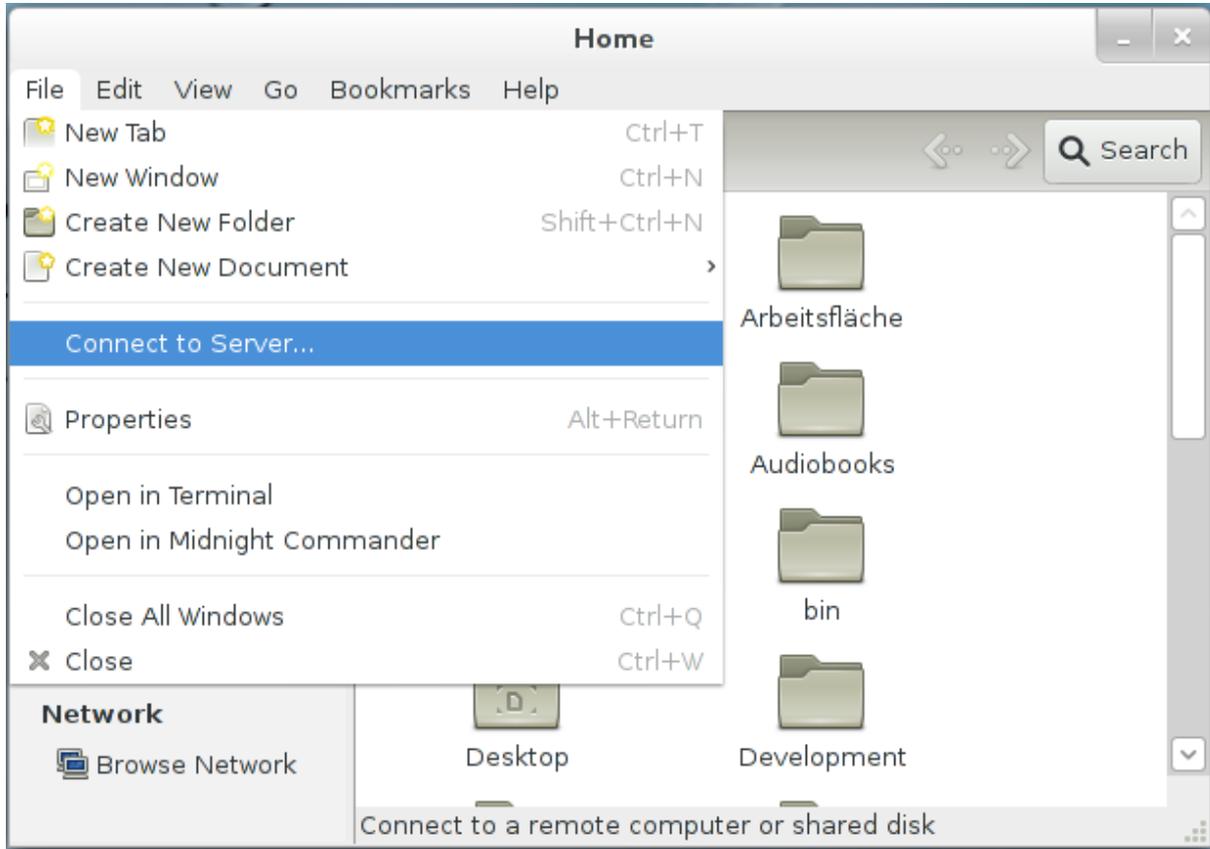
- That's it, now the databases can be browsed:



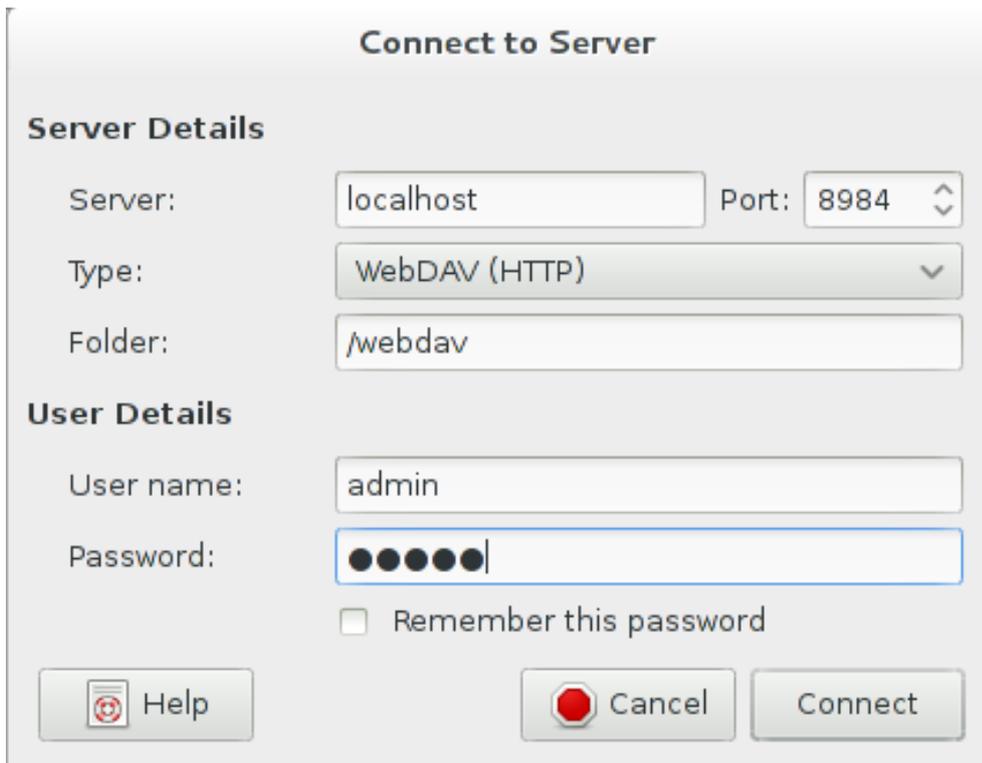
WebDAV: GNOME

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with GNOME and Nautilus.

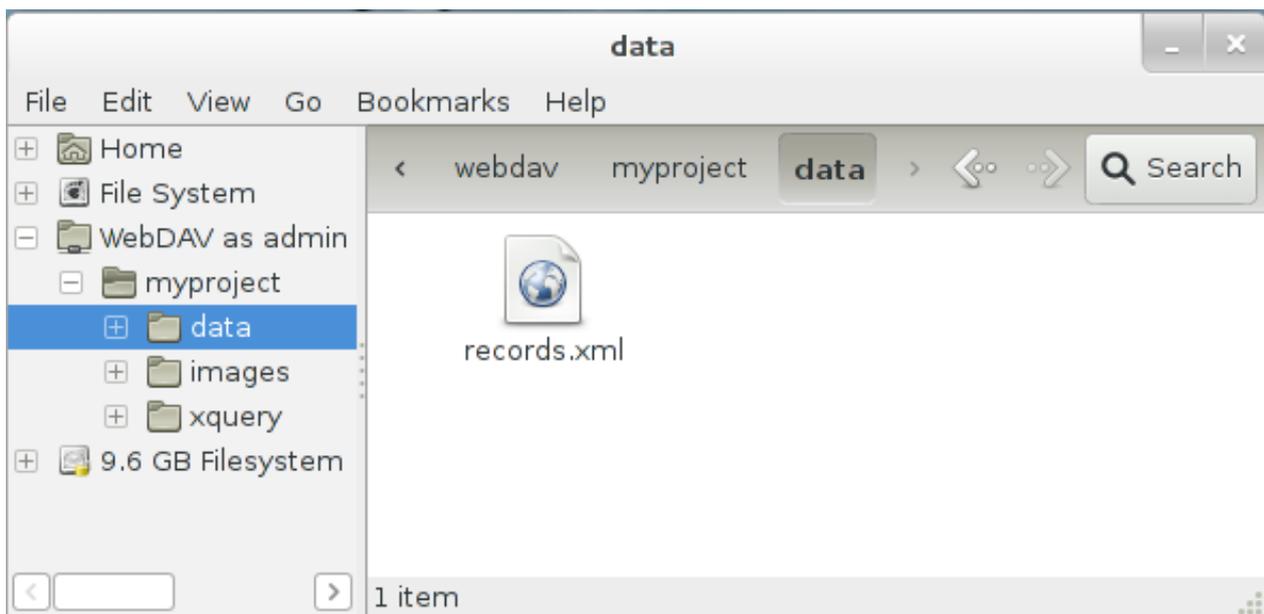
- In Nautilus choose File -> Connect to Server:



- Choose "WebDAV (HTTP)" from the "Type" drop-down and enter the server address, port and user credentials:



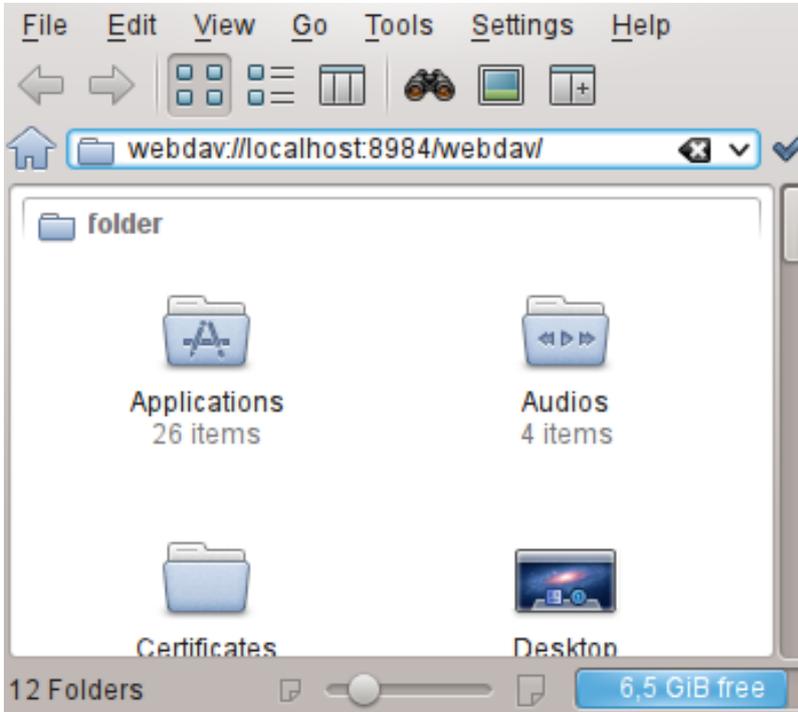
- After clicking "Connect" the databases can be browsed:



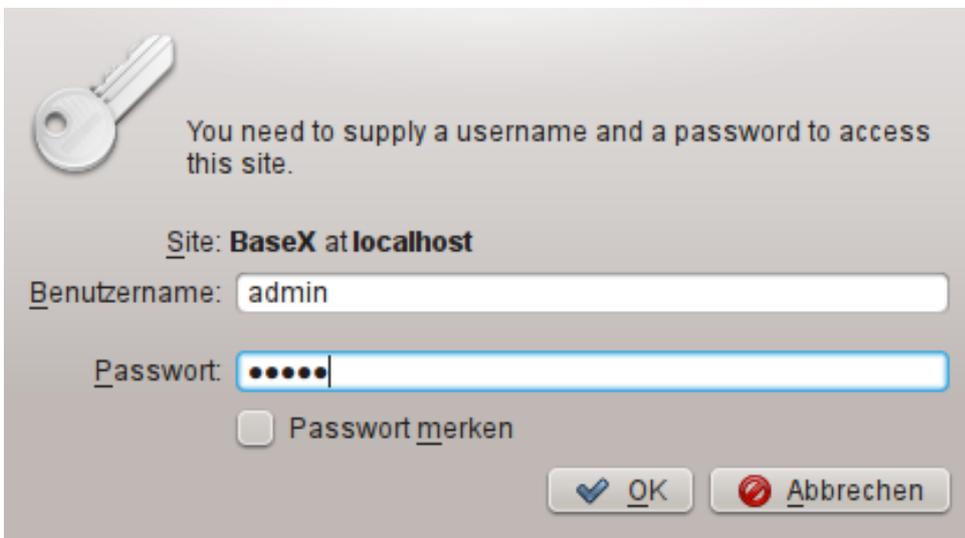
WebDAV: KDE

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with KDE.

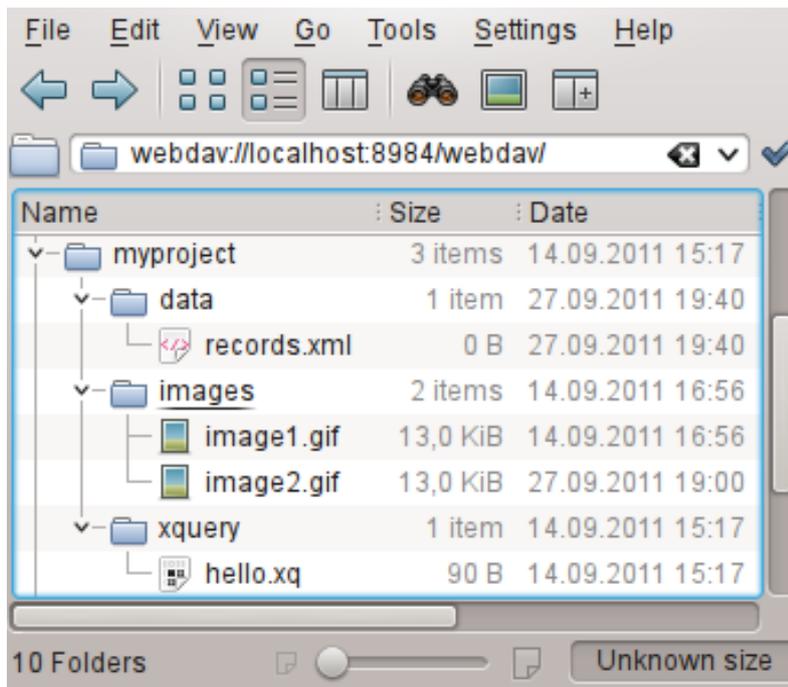
- KDE SC provides two file managers - Dolphin and Konqueror, which both support WebDAV using the "webdav://" URL prefix. Start Dolphin or Konqueror and enter the BaseX WebDAV URL (eg. webdav://localhost:8984/webdav):



- Enter the user credentials:



- After clicking "OK" the databases can be browsed:



Client APIs

Clients

This page is part of the Developer Section. It describes how to use BaseX from other programming languages.

With the following light-weight bindings in different programming languages, you will be able to connect to a running BaseX server instance, execute database commands, perform queries or listen to events. Most clients offer the following two classes:

- Standard Mode: connecting to a server, sending commands
- Query Mode: defining queries, binding variables, iterative evaluation

Please have a look at our Server Protocol for more information on the clients and the underlying protocol.

Bindings for other languages are easy to write; your contributions are welcome.

Currently, we offer bindings for the following programming languages:

Object oriented

C# ^[1], VB ^[2], Scala ^[3], Java ^[4]

ActionScript ^[5] (contributed by Manfred Knobloch)

Scripting

Perl ^[6], Ruby ^[7]

PHP ^[8] (example)

Python ^[9] (improved by Arjen van Elteren)

Rebol ^[10] (contributed by Sabu Francis)

Functional

Lisp ^[11] (contributed by Andy Chambers)

Haskell ^[12] (contributed by Leo Wörteler)

Others

node.js ^[13] (contributed by Andy Bunce)

Qt ^[14] (contributed by Hendrik Strobelt)

C ^[15]

Many of the interfaces contain the following files:

- `BaseXClient` contains the code for creating a session, sending and executing commands and receiving results. An inner `Query` class facilitates the binding of external variables and iterative query evaluation.
 - `Example` demonstrates how to send database commands.
 - `QueryExample` shows you how to evaluate queries in an iterative manner.
 - `QueryBindExample` shows you how to bind a variable to your query and evaluates the query in an iterative manner.
 - `CreateExample` shows how new databases can be created by using streams.
 - `AddExample` shows how documents can be added to a database by using streams.
 - `EventExample` demonstrates how to watch and unwatch Events.
-

References

- [1] <https://github.com/BaseXdb/basex-api/tree/master/src/main/c%23>
- [2] <https://github.com/BaseXdb/basex-api/tree/master/src/main/vb>
- [3] <https://github.com/BaseXdb/basex-api/tree/master/src/main/scala>
- [4] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples/api>
- [5] <https://github.com/BaseXdb/basex-api/tree/master/src/main/as>
- [6] <https://github.com/BaseXdb/basex-api/tree/master/src/main/perl>
- [7] <https://github.com/BaseXdb/basex-api/tree/master/src/main/ruby>
- [8] <https://github.com/BaseXdb/basex-api/tree/master/src/main/php>
- [9] <https://github.com/BaseXdb/basex-api/tree/master/src/main/python>
- [10] <https://github.com/BaseXdb/basex-api/tree/master/src/main/rebol>
- [11] <https://github.com/BaseXdb/basex-api/tree/master/src/main/lisp>
- [12] <https://github.com/BaseXdb/basex-api/tree/master/src/main/haskell>
- [13] <https://github.com/apb2006/basex-node>
- [14] <https://github.com/BaseXdb/basex-api/tree/master/src/main/qt>
- [15] <https://github.com/BaseXdb/basex-api/tree/master/src/main/c>

Standard Mode

In the standard mode of the Clients, a database command can be sent to the server using the `execute()` function of the `Session`. This function returns the whole result. With the `info()` function, you can request some information on your executed process. If an error occurs, an exception with the error message will be thrown.

Usage

The standard execution works as follows:

1. Create a new session instance with hostname, port, username and password.
2. Call the `execute()` function of the session with the database commands as argument.
3. Receive the result of a successfully executed command. If an error occurs, an exception is thrown.
4. Optionally, call `info()` to get some process information
5. Continue using the client (back to 2.), or close the session.

Example in PHP

Taken from our repository ^[1]:

```
<?php
/*
 * This example shows how database commands can be executed.
 * Documentation: http://basex.org/api
 *
 * (C) BaseX Team 2005-11, BSD License
 */
include("BaseXClient.php");

try {
    // initialize timer
    $start = microtime(true);

    // create session
```

```
$session = new Session("localhost", 1984, "admin", "admin");

// perform command and print returned string
print $session->execute("xquery 1 to 10");

// close session
$session->close();

// print time needed
$time = (microtime(true) - $start) * 1000;
print "\n$time ms\n";

} catch (Exception $e) {
    // print exception
    print $e->getMessage();
}
?>
```

References

[1] <https://github.com/BaseXdb/basex-api/blob/master/src/main/php/Example.php>

Query Mode

The query mode of the Clients allows you to bind external variables to a query and evaluate the query in an iterative manner. The `query()` function of the `Session` instance returns a new query instance.

Usage

The query execution works as follows:

1. Create a new session instance with hostname, port, username and password.
2. Call `query()` with your XQuery expression to get a query object.
3. Optionally bind variables to the query with one of the `bind()` functions.
4. **Version 7.2:** Optionally bind a value to the context item via `context()`.
5. Iterate through the query object with the `more()` and `next()` functions.
6. As an alternative, call `execute()` to get the whole result at a time.
7. `info()` gives you information on query evaluation.
8. `options()` returns the query serialization parameters.
9. Don't forget to close the query with `close()`.

PHP Example

Taken from our repository ^[1]:

```
<?php
/*
 * This example shows how queries can be executed in an iterative manner.
 * Documentation: http://basex.org/api
```

```
*
* (C) BaseX Team 2005-11, BSD License
*/
include("BaseXClient.php");

try {
    // create session
    $session = new Session("localhost", 1984, "admin", "admin");

    try {
        // create query instance
        $input = 'declare variable $name external; '.
            'for $i in 1 to 10 return element { $name } { $i }';
        $query = $session->query($input);

        // bind variable
        $query->bind("$name", "number");

        // print result
        print $query->execute()."\n";

        // close query instance
        $query->close();

    } catch (Exception $e) {
        // print exception
        print $e->getMessage();
    }

    // close session
    $session->close();

} catch (Exception $e) {
    // print exception
    print $e->getMessage();
}
?>
```

References

- [1] <https://github.com/BaseXdb/basex-api/blob/master/src/main/php/QueryBindExample.php>

PHP Example

This page is referenced from the Clients page. It demonstrates how database commands and XQuery can be executed on a database server with a PHP client. The results of the query are stored into a DOM document and can be processed in several ways.

Requirements

- BaseXClient ^[1]: BaseX PHP Client
- DOMExample ^[2]: Example used in this tutorial
- any PHP Server, such as XAMPP ^[3]

Setting up

- Install and start XAMPP, or choose a PHP Server of your own
- Copy `BaseXClient.php` and `DOMExample.php` to the XAMPP folder or upload it to your webserver

Usage

1. Start a Database Server instance on your local or a remote machine. Make sure the `host` and `port` settings in `DOMExample.php` are correct.
2. Call `DOMExample.php` in a web browser of your choice.
3. Look at the DOM document ^[4] on the PHP documentation for further information on the DOM document functions.
4. Open `DOMExample.php` in an editor and edit it for your own needs.

References

- [1] <https://github.com/BaseXdb/basex-api/blob/master/src/main/php/BaseXClient.php>
 - [2] <https://github.com/BaseXdb/basex-api/blob/master/src/main/php/DOMExample.php>
 - [3] <http://www.apachefriends.org/en/xampp.html>
 - [4] <http://php.net/manual/en/class.domdocument.php>
-

Server Protocol

This page presents the classes and functions of the BaseX Clients, and the underlying protocol, which is utilized for communicating with the database server.

Workflow

- First of all, a BaseX database server must be running, which will process the client requests.
- Each client provides a session class or script with methods to connect to and communicate with the database server. A socket connection will be established by the constructor, which expects a host, port, user name and password as arguments.
- The `execute()` method is called to launch a database command. It returns the result or throws an exception with the received error message.
- The `query()` method creates a query instance. Variables can be bound to that object, and the result can either be requested via `execute()`, or in an iterative manner with the `more()` and `next()` functions. If an error occurs, an exception will be thrown.
- The `create()`, `add()`, `replace()` and `store()` method pass on input streams to the corresponding database commands.
- To speed up execution, an output stream can be specified by some clients; this way, all results will be directed to that output stream.
- Most clients are accompanied by some example files, which demonstrate how database commands can be executed or how queries can be evaluated.

Class Structure

Session

- Create and return session with host, port, user name and password:
`Session(String host, int port, String name, String password)`
 - Execute a command and return the result:
`String execute(String command)`
 - Return a query object for the specified query:
`Query query(String query)`
 - Create a database from an input stream:
`void create(String name, InputStream in)`
 - Add a document to the current database from an input stream:
`void add(String path, InputStream in)`
 - Replace a document with the specified input stream:
`void replace(String path, InputStream in)`
 - Store raw data at the specified path:
`void store(String path, InputStream in)`
 - Watch the specified event:
`void watch(String name, Event notifier)`
 - Unwatch the specified event:
`void unwatch(String name)`
-

- Return process information:

```
String info()
```

- Close the session:

```
void close()
```

Query

- Create query object with session and query:

```
Query(Session s, String query)
```

- Bind an external variable:

```
void bind(String name, String value, String type). The type can be an empty string.
```

- **Version 7.2:** Bind the context item:

```
void context(String value, String type). The type can be an empty string.
```

- Execute the query and return the result:

```
String execute()
```

- Iterator: check if a query returns more items:

```
boolean more()
```

- Iterator: return the next item:

```
String next()
```

- Return query information:

```
String info()
```

- Return serialization parameters:

```
String options()
```

- Return if the query may perform updates:

```
boolean updating()
```

- Close the query:

```
void close()
```

Transfer Protocol

All Clients use the following client/server protocol to communicate with the server. The description of the protocol is helpful if you want to implement a new client.

General Syntax

- `\x`: single byte.
- `{...}`: utf8 strings or raw data, suffixed with a `\0` byte. To avoid confusion with the suffix byte, all `\0` and `\xFF` bytes that occur in raw data will be prefixed with `\xFF`.

Authentication (via cram-md5 ^[1])

1. Client connects to server socket
2. Server sends timestamp:
{timestamp}
3. Client sends username and hashed password/timestamp:
{username} {md5(md5(password) + timestamp)}
4. Server replies with \0 (success) or \1 (error)

Command Protocol

The following byte sequences are sent and received from the client (please note that a specific client need not support all of the presented commands):

Command	Client Request	Server Response	Description
COMMAND	{command}	{result} {info} \0	Executes a database command.
QUERY	\0 {query}	{id} \0	Creates a new query instance and returns its id.
CREATE	\8 {name} {input}	{info} \0	Creates a new database with the specified input (may be empty).
ADD	\9 {name} {path} {input}	{info} \0	Adds a new resource to the opened database.
WATCH	\10 {name}		Registers the client for the specified event.
UNWATCH	\11 {name}		Unregisters the client.
REPLACE	\12 {path} {input}	{info} \0	Replaces a resource with the specified input.
STORE	\13 {path} {input}	{info} \0	Stores a binary resource in the opened database.
␣ error		{beginning of result} {error} \1	Error feedback.

Query Command Protocol

Queries are referenced via an `id`, which has been returned by the `QUERY` command (see above):

Query Command	Client Request	Server Response	Description
CLOSE	\2 {id}	\0 \0	Closes and unregisters the query with the specified id.
BIND	\3 {id} {name} {value} {type}	\0 \0	Binds a value to a variable. An empty string can be specified as data type.
RESULTS	\4 {id}	\x {item} ... \x {item} \0	Returns all resulting items as strings, prefixed by a Type byte (\x).
EXECUTE	\5 {id}	{result} \0	Executes the query and returns all results as a UTF8 string.
INFO	\6 {id}	{result} \0	Returns a query info string.
OPTIONS	\7 {id}	{result} \0	Returns a string with all query serialization parameters.
CONTEXT	\14 {id} {value} {type}	\0 \0	Version 7.2: Binds a value to the context item. An empty string can be specified as data type.
UPDATING	\30 {id}	{result} \0	Version 7.2: Returns <code>true</code> if the query may perform updates; <code>false</code> otherwise.
FULL	\31 {id}	XDM {item} ... XDM {item} \0	Version 7.2: Returns all resulting items as strings, prefixed by XDM Meta Data (see below).

␣ error		{beginning of result} \1 {error}	Error feedback.
---------	--	-------------------------------------	-----------------

XDM Meta Data

In most cases, the *XDM* meta data boils down to a single byte, which represents the node kind or item type. There are three exceptions: `document-node()`, `attribute()` and `xs:QName` items are followed by an additional `{URI}` string.

Examples

- Java client ^[2]
- C# client ^[3]
- Python client ^[4]
- Perl client ^[5]

Changelog

Version 7.2

- Added: Query Commands CONTEXT, UPDATING and FULL

References

- [1] <http://tools.ietf.org/html/rfc2195>
- [2] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/api/BaseXClient.java>
- [3] <https://github.com/BaseXdb/basex-api/blob/master/src/main/c%23/BaseXClient.cs>
- [4] <https://github.com/BaseXdb/basex-api/blob/master/src/main/python/BaseXClient.py>
- [5] <https://github.com/BaseXdb/basex-api/blob/master/src/main/perl/BaseXClient.pm>

Server Protocol: Types

The following table lists all node kinds and item types that are returned by the Server Protocol. Currently, all node kinds are of type `xs:untypedAtomic`:

ID	Node Kind/Item Type	Type
7	Function item	<i>function</i>
8	<code>node()</code>	<i>node</i>
9	<code>text()</code>	<i>node</i>
10	<code>processing-instruction()</code>	<i>node</i>
11	<code>element()</code>	<i>node</i>
12	<code>document-node()</code>	<i>node</i>
13	<code>document-node(element())</code>	<i>node</i>
14	<code>attribute()</code>	<i>node</i>
15	<code>comment()</code>	<i>node</i>
32	<code>item()</code>	<i>atomic value</i>
33	<code>xs:untyped</code>	<i>atomic value</i>
34	<code>xs:anyType</code>	<i>atomic value</i>
35	<code>xs:anySimpleType</code>	<i>atomic value</i>
36	<code>xs:anyAtomicType</code>	<i>atomic value</i>
37	<code>xs:untypedAtomic</code>	<i>atomic value</i>
38	<code>xs:string</code>	<i>atomic value</i>
39	<code>xs:normalizedString</code>	<i>atomic value</i>
40	<code>xs:token</code>	<i>atomic value</i>
41	<code>xs:language</code>	<i>atomic value</i>
42	<code>xs:NMTOKEN</code>	<i>atomic value</i>
43	<code>xs>Name</code>	<i>atomic value</i>
44	<code>xs:NCName</code>	<i>atomic value</i>
45	<code>xs:ID</code>	<i>atomic value</i>
46	<code>xs:IDREF</code>	<i>atomic value</i>
47	<code>xs:ENTITY</code>	<i>atomic value</i>
48	<code>xs:float</code>	<i>atomic value</i>
49	<code>xs:double</code>	<i>atomic value</i>
50	<code>xs:decimal</code>	<i>atomic value</i>
51	<code>xs:precisionDecimal</code>	<i>atomic value</i>
52	<code>xs:integer</code>	<i>atomic value</i>
53	<code>xs:nonPositiveInteger</code>	<i>atomic value</i>
54	<code>xs:negativeInteger</code>	<i>atomic value</i>
55	<code>xs:long</code>	<i>atomic value</i>
56	<code>xs:int</code>	<i>atomic value</i>

57	xs:short	<i>atomic value</i>
58	xs:byte	<i>atomic value</i>
59	xs:nonNegativeInteger	<i>atomic value</i>
60	xs:unsignedLong	<i>atomic value</i>
61	xs:unsignedInt	<i>atomic value</i>
62	xs:unsignedShort	<i>atomic value</i>
63	xs:unsignedByte	<i>atomic value</i>
64	xs:positiveInteger	<i>atomic value</i>
65	xs:duration	<i>atomic value</i>
66	xs:yearMonthDuration	<i>atomic value</i>
67	xs:dayTimeDuration	<i>atomic value</i>
68	xs:dateTime	<i>atomic value</i>
69	xs:dateTimeStamp	<i>atomic value</i>
70	xs:date	<i>atomic value</i>
71	xs:time	<i>atomic value</i>
72	xs:gYearMonth	<i>atomic value</i>
73	xs:gYear	<i>atomic value</i>
74	xs:gMonthDay	<i>atomic value</i>
75	xs:gDay	<i>atomic value</i>
76	xs:gMonth	<i>atomic value</i>
77	xs:boolean	<i>atomic value</i>
78	base64:binary	<i>atomic value</i>
79	xs:base64Binary	<i>atomic value</i>
80	xs:hexBinary	<i>atomic value</i>
81	xs:anyURI	<i>atomic value</i>
82	xs:QName	<i>atomic value</i>
83	xs:NOTATION	<i>atomic value</i>

Java Examples

This page is part of the Developer Section. The following Java code snippets demonstrate how easy it is to run database commands, create collections, perform queries, etc. via the BaseX API. Most examples are taken from our [basex-examples](#)^[1] repository, in which you will find some more use cases.

Since 7.2.1, our XQJ API is based on Charles Foster's XQJ implementation^[2], which fully utilized the client/server architecture of BaseX. Note that the older XML:DB API can only be used in embedded mode.

Local Examples

`RunCommands.java`^[3]

creates and drops database and index instances, prints a list of all existing databases.

`WikiExample.java`^[4]

creates a database from an url (wiki instance), runs a query against it and drops the database.

`RunQueries.java`^[5]

shows three variants of running queries.

`CreateCollection.java`^[6]

creates and manages a collection.

`QueryCollection.java`^[7]

creates, runs queries against it and drops a collection.

Server Examples

`ServerCommands.java`^[8]

launches server-side commands using a client session.

`ServerAndLocal.java`^[9]

processes server results locally.

`ServerConcurrency.java`^[10]

runs concurrent queries.

`UserExample.java`^[11]

manages database users.

REST API Examples

`RESTGet.java`^[12]

presents the HTTP GET method.

`RESTPost.java`^[13]

presents the HTTP POST method.

`RESTPut.java`^[14]

presents the HTTP PUT method.

`RESTDelete.java`^[15]

presents the HTTP DELETE method.

XML:DB API Examples

XMLDBCreate.java ^[16]

creates a collection using XML:DB.

XMLDBQuery.java ^[17]

runs a query using XML:DB.

XMLDBInsert.java ^[18]

inserts a document into a database using XML:DB.

References

- [1] <https://github.com/BaseXdb/basex-examples/tree/master/src/main/java/org/basex/examples>
- [2] <http://xqj.net/basex/>
- [3] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/query/RunCommands.java>
- [4] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/WikiExample.java>
- [5] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/query/RunQueries.java>
- [6] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/query/CreateCollection.java>
- [7] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/query/QueryCollection.java>
- [8] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/server/ServerCommands.java>
- [9] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/server/ServerAndLocal.java>
- [10] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/server/ServerConcurrency.java>
- [11] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/UserExample.java>
- [12] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/rest/RESTGet.java>
- [13] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/rest/RESTPost.java>
- [14] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/rest/RESTPut.java>
- [15] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/rest/RESTDelete.java>
- [16] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/xmldb/XMLDBCreate.java>
- [17] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/xmldb/XMLDBQuery.java>
- [18] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/xmldb/XMLDBInsert.java>

Advanced User's Guide

Advanced User's Guide

This page is one of the Main Sections of the documentation. It contains details on the BaseX storage and the Server architecture, and presents some more GUI features.

Storage

- Configuration: BaseX start files and directories
- Indexes: Available index structures and their utilization
- Backups: Backup and restore databases
- Catalog Resolver Information on entity resolving
- Statistics: Exemplary statistics on databases created with BaseX
- Storage Layout: How data is stored in the database files

Server and Query Architecture

- User Management: User management in the client/server environment
 - Transaction Management: Insight into the BaseX transaction management
 - Logging: Description of the server logs
 - Events: Description of the event feature
 - Execution Plan: Analyzing query evaluation
-

Configuration

This article is part of the Advanced User's Guide. It gives some more insight into the configuration of BaseX.

Configuration Files

BaseX maintains some configuration files, which are stored in the project's Home Directory:

- `.basex` contains all options that are relevant for running the server or standalone versions of BaseX.
- `.basexgui` defines all options relevant to the BaseX GUI.
- `.basexperm` contains user name, passwords, and permissions (see last paragraph).
- `.basexevents` contains all existing events (see Events).
- `.basexhistory` contains commands that have been typed in most recently.

Note that, depending on your OS and configuration, files and folders with a '.' prefix may be hidden.

Home Directory

As BaseX is distributed in different flavors, and may be started from different locations, it dynamically determines its home directory:

- First, the **system property** `"org.basex.path"` is checked. If it contains a value, it is chosen as directory path.
- If not, the **current user directory** (defined by the system property `"user.dir"`) is chosen if the `.basex` configuration file is found in this directory.
- Otherwise, the configuration file is searched in the **application directory** (the folder in which the project is located).
- In all other cases, the **user's home directory** (defined in `"user.home"`) is chosen.

Database Directory

A database in BaseX consists of several files, which are all bundled in a folder with the database name. If the user's home directory has been chosen as base directory, the database folders will be stored in a `BaseXData` directory. Otherwise, the directory will be named `datda`.

The database path can be changed as follows:

- GUI: Choose *Options* → *Preferences* and choose a new database path.
- Server/Standalone: `SET DBPATH [path]`

Note: Existing databases will not be automatically moved to the new destination.

User and Log Files

Global users, along with their passwords and permissions, are stored in the `.basexperm` file in the home directory. Local users and permissions are stored inside the database files. Log files are stored in text format in the home directory `BaseXData/.logs` (see Logging for more information).

Indexes

This article is part of the Advanced User's Guide and introduces the available index structures, which are utilized by the query optimizer to rewrite expressions and speed up query evaluation.

Nearly all examples in this article are based on the `factbook.xml` ^[3] document. To see how a query is rewritten, please turn on the Info View in the GUI or use the `-V` flag on command line.

Structural Indexes

Structural indexes will always be present and cannot be dropped by the user:

Name Index

The name index contains all element and attribute names of a database, and the fixed-size index ids are stored in the main database table. If a database is updated, new names are automatically added. The index is further enriched with statistical information, which will get out-of-dated after new updates.

The name index is applied to pre-evaluate location steps that will never yield results:

```
(: will be rewritten to an empty sequence :)  
/non-existing-name
```

The contents of the name indexes can be directly accessed via the XQuery functions `index:element-names()` and `index:attribute-names()`.

Path Index

The path index (also called *path summary*) stores all distinct paths of the documents in the database. It also contains additional statistical information. Currently, the index will get out-of-dated after new updates.

The path index is applied to rewrite descendant steps to multiple child steps. Child steps can be evaluated faster, as less nodes have to be accessed:

```
doc('factbook.xml')//province,  
(: ...will be rewritten to... :)  
doc('factbook.xml')/mondial/country/province
```

The paths statistics are e.g. used to pre-evaluate the `count()` function:

```
(: will be rewritten and pre-evaluated by the path index :)  
count(doc('factbook')//country)
```

The contents of the path index can be directly accessed via the XQuery function `index:facets()`.

Document Index

The document index contains references to the `pre` values of all document nodes. It speeds up the access to specific documents in a database, and it will be automatically updated when updates are performed.

The following query will be sped up by the document index:

```
db:open('DatabaseWithLotsOfDocuments')
```

Value Indexes

Value indexes can be optionally created and dropped by the user. The text and attribute index will be created by default:

Text Index

This index speeds up string-based equality tests on text nodes. The `UPDINDEX` option can be activated to keep this index up-to-date.

The following queries will all be rewritten for index access:

```
(: 1st example :)
/*[text() = 'Germany'],
(: 2nd example :)
doc('factbook.xml')//name[. = 'Germany'],
(: 3rd st example :)
for $c in db:open('factbook')//country
where $c//city/name = 'Hanoi'
return $c/name
```

Text nodes can be directly accessed from the index via the XQuery function `db:text()`. The contents of the index can be accessed via `index:texts()`.

Since [Version 7.2.1](#), the text index also supports range queries based on string comparisons:

```
(: 1st example :)
db:open('Library')//Medium[Year >= '2005' and Year <= '2007'],
(: 2nd example :)
let $min := '2011-04-16T00:00:00'
let $max := '2011-04-19T23:59:59'
return db:open('news')//entry[date-time > $min and date-time < $max]
```

Text nodes can be directly accessed from the index via the XQuery function `db:text-range()`.

Attribute Index

Similar to the text index, this index speeds up string-based equality and range tests on attribute values. The `UPDINDEX` option can be activated to keep this index up-to-date.

The following queries will all be rewritten for index access:

```
(: 1st example :)
//country[@car_code = 'J'],
(: 2nd example :)
//province[@* = 'Hokkaido']//name,
```

```
(: 3rd example :)  
//sea[@depth > '2100' and @depth < '4000']
```

Text nodes can be directly accessed from the index via the XQuery functions `db:attribute()` and `db:attribute-range()`. The contents of the index can be accessed via `index:attributes()`.

Full-Text Index

The Full-Text index speeds up queries using the `contains text` expression. Internally, two index structures are provided: the default index sorts all keys alphabetically by their character length. It is particularly fast if fuzzy searches are performed. The second index is a compressed trie structure, which needs slightly more memory, but is specialized on wildcard searches. Both index structures will be merged in a future version of BaseX.

The following queries are examples for expressions that will be optimized for index access (provided that the relevant index exists in a particular database):

If the full-text index exists, the following queries will all be rewritten for index access:

```
(: 1st example :)  
//country/name[text() contains text 'and'],  
(: 2nd example :)  
//religions[. contains text { 'Catholic', 'Roman' }  
  using case insensitive distance at most 2 words]
```

Text nodes can be directly accessed from the index via the XQuery function `db:fulltext()`. The Full-Text Module contains additional functions for retrieving index data.

Backups

This page is part of the Advanced User's Guide. The following two paragraphs demonstrate how to create a backup and restore the database within BaseX.

GUI Example

1. Start the BaseX GUI and create a new database in *Database* → *New...* with your XML document.
2. Go to *Database* → *Manage...* and create a backup of your database. The backup will be created in the database directory.
3. Go to *Database* → *Add...* and add another document.
4. Go to *Database* → *Manage...* and restore your database. The database will be restored from the latest backup of to the database found in the database directory.

Console Example

1. Start the BaseX Standalone client from a console.
2. Create a new database via the CREATE DB command.
3. Use the CREATE BACKUP command to back up your database.
4. Add a new document via ADD: `ADD AS newdoc.xml <newdoc/>`
5. Use the RESTORE command to restore the original database.
6. Type in XQUERY / to see the restored database contents.

The same commands can be used with a BaseX client connected to a remote Database Server.

Catalog Resolver

This article is part of the Advanced User's Guide. It clarifies how to deal with external DTD declarations when parsing XML data.

Overview

XML documents often rely on Document Type Definitions (DTDs). While parsing a document with BaseX, entities can be resolved with respect to that particular DTD. By default, the DTD is only used for entity resolution.

XHTML, for example, defines its doctype via the following line:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Fetching `xhtml1-strict.dtd` obviously involves network traffic. When dealing with single files, this may seem tolerable, but importing large collections benefits from caching these resources. Depending on the remote server, you will experience significant speed improvements when caching DTDs locally.

XML Entity and URI Resolvers

BaseX comes with a default URI resolver that is usable out of the box.

To enable entity resolving you have to provide a valid XML Catalog file, so that the parser knows where to look for mirrored DTDs.

A simple working example for XHTML might look like this:

```
<?xml version="1.0"?>
<catalog prefer="system" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteSystem systemIdStartString="http://www.w3.org/TR/xhtml1/DTD/" rewritePrefix="file:///path/to/dtds/" />
</catalog>
```

This rewrites all systemIds starting with: `http://www.w3.org/TR/xhtml1/DTD/` to `file:///path/to/dtds/`.

The XHTML DTD `xhtml1-strict.dtd` and all its linked resources will now be loaded from the specified path.

GUI Mode

When running BaseX in GUI mode, simply provide the path to your XML Catalog file in the *Parsing* Tab of the Database Creation Dialog.

Console & Server Mode

To enable Entity Resolving in Console Mode, specify the following options:

- `SET CATFILE [path]`

Now entity resolving is active for the current session. All subsequent `ADD` commands will use the catalog file to resolve entities.

The **paths** to your catalog file and the actual DTDs are either absolute or relative to the *current working directory*. When using BaseX in Client-Server-Mode, this is relative to the *server's* working directory.

Please Note

Entity resolving only works if the internal XML parser is switched off (which is the default case). If you use the internal parser, you can manually specify whether you want to parse DTDs and entities or not.

Using other Resolvers

There might be some cases when you do not want to use the built-in resolver that Java provides by default (via `com.sun.org.apache.xml.internal.resolver.*`).

BaseX offers support for the Apache-maintained XML Commons Resolver ^[1], available for download here ^[2].

To use it add **resolver.jar** to the classpath when starting BaseX:

```
java -cp basex.jar:resolver.jar org.basex.BaseXServer
```

More Information

- Wikipedia on Document Type Definitions ^[3]
- Apache XML Commons Article on Entity Resolving ^[4]
- XML Entity and URI Resolvers ^[5], Sun
- XML Catalogs. OASIS Standard, Version 1.1. 07-October-2005. ^[6]

References

[1] <http://xml.apache.org/commons>

[2] <http://xerces.apache.org/mirrors.cgi>

[3] http://en.wikipedia.org/wiki/Document_Type_Definition

[4] <http://xml.apache.org/commons/components/resolver/resolver-article.html>

[5] <http://java.sun.com/webservices/docs/1.6/jaxb/catalog.html>

[6] <http://www.oasis-open.org/committees/download.php/14810/xml-catalogs.pdf>

Statistics

This article is part of the Advanced User's Guide. It lists statistics on various XML instances that have been created with BaseX. The URLs to the original sources, if available or public, are listed below.

Databases

- FileSize is the original size of the input documents
- #Files indicates the number of stored XML documents
- #DbSize is the size of the resulting database (excluding the value index structures)
- #Nodes represents the number of XML nodes (elements, attributes, texts, etc.) stored in the database
- #Attr indicates the maximum number of attributes stored for a single element
- #ENames and #ANames reflect the number of distinct element and attribute names
- #URIs represent the number of distinct namespace URIs
- Height indicates the maximum level depth of the stored nodes

If a fixed database limit is reached, documents can be distributed in several database instances, which can then be accessed from a single XQuery expression.

Instances	FileSize	#Files	DbSize	#Nodes	#Attr	#ENames	#ANames	#URIs	Height
Limits	512 GiB (2 ³⁹ Bytes)	536'870'912 (2 ²⁹)	<i>no limit</i>	2'147'483'648 (2 ³¹)	<i>no limit</i>	32768 (2 ¹⁵)	32768 (2 ¹⁵)	256 (2 ⁸)	<i>no limit</i>
RuWikiHist	421 GiB	1	416 GiB	324'848'508	3	21	6	2	6
ZhWikiHist	126 GiB	1	120 GiB	179'199'662	3	21	6	2	6
EnWiktionary	79 GiB	1	75 GiB	134'380'393	3	21	6	2	6
XMark	55 GiB	1	64 GiB	1'615'071'348	2	74	9	0	13
EnWikiMeta	54 GiB	1	52 GiB	401'456'348	3	21	6	2	6
MedLine	38 GiB	379	36 GiB	1'623'764'254	2	84	6	0	9
iProClass	36 GiB	1	37 GiB	1'631'218'984	3	245	4	2	9
Inex209	31 GiB	2'666'500	34 GiB	1'336'110'639	15	28'034	451	1	37
CoPhIR	29 GiB	10'000'000	31 GiB	1'104'623'376	10	42	42	0	8
EnWikipedia	26 GiB	1	25 GiB	198'546'747	3	24	21	2	6
XMark	22 GiB	1	26 GiB	645'997'965	2	74	9	0	13
InterPro	14 GiB	1	19 GiB	860'304'235	5	7	15	0	4
GenomeI	13 GiB	1	13 GiB	432'628'105	12	26	101	2	6
NewYorkTimes	12 GiB	1'855'659	13 GiB	280'407'005	5	41	33	0	6
TrEMBL	11 GiB	1	14 GiB	589'650'535	8	47	30	2	7
XMark	11 GiB	1	13 GiB	323'083'409	2	74	9	0	13
IntAct	7973 MiB	25'624	6717 MiB	297'478'392	7	64	22	2	14
Freebase	7366 MiB	1	10 GiB	443'627'994	8	61	283	1	93
SDMX	6356 MiB	1	8028 MiB	395'871'872	2	22	6	3	7
OpenStreetMap	5312 MiB	1	5171 MiB	6'910'669	3	19	5	2	6
SwissProt	4604 MiB	1	5422 MiB	241'274'406	8	70	39	2	7
EURLex	4815 MiB	1	5532 MiB	167'328'039	23	186	46	1	12

Wikicorpus	4492 MiB	659'338	4432 MiB	157'948'561	12	1'257	2'687	2	50
EnWikiRDF	3679 MiB	1	3537 MiB	98'433'194	1	11	2	11	4
CoPhIR	2695 MiB	1'000'000	2882 MiB	101'638'857	10	42	42	0	8
MeSH	2091 MiB	1	2410 MiB	104'845'819	3	6	5	2	5
FreeDB	1723 MiB	1	2462 MiB	102'901'519	2	7	3	0	4
XMark	1134 MiB	1	1303 MiB	32'298'989	2	74	9	0	13
DeepFS	810 MiB	1	850 MiB	44'821'506	4	3	6	0	24
LibraryUKN	760 MiB	1	918 MiB	46'401'941	3	23	3	0	5
Twitter	736 MiB	1'177'495	767 MiB	15'309'015	0	8	0	0	3
Organizations	733 MiB	1'019'132	724 MiB	33'112'392	3	38	9	0	7
DBLP	694 MiB	1	944 MiB	36'878'181	4	35	6	0	7
Feeds	692 MiB	444'014	604 MiB	5'933'713	0	8	0	0	3
MedLineSupp	477 MiB	1	407 MiB	21'602'141	5	55	7	0	9
AirBase	449 MiB	38	273 MiB	14'512'851	1	111	5	0	11
MedLineDesc	260 MiB	1	195 MiB	10'401'847	5	66	8	0	9
ZDNET	130 MiB	95'663	133 MiB	3'060'186	21	40	90	0	13
JMNEdict	124 MiB	1	171 MiB	8'592'666	0	10	0	0	5
XMark	111 MiB	1	130 MiB	3'221'926	2	74	9	0	13
Freshmeat	105 MiB	1	86 MiB	3'832'028	1	58	1	0	6
DeepFS	83 MiB	1	93 MiB	4'842'638	4	3	6	0	21
Treebank	82 MiB	1	92 MiB	3'829'513	1	250	1	0	37
DBLP2	80 MiB	170'843	102 MiB	4'044'649	4	35	6	0	6
DDI	76 MiB	3	39 MiB	2'070'157	7	104	16	21	11
Alfred	75 MiB	1	68 MiB	3'784'285	0	60	0	0	6
University	56 MiB	6	66 MiB	3'468'606	1	28	4	0	5
MediaUKN	38 MiB	1	45 MiB	1'619'443	3	21	3	0	5
HCIBIB2	32 MiB	26'390	33 MiB	617'023	1	39	1	0	4
Nasa	24 MiB	1	25 MiB	845'805	2	61	8	1	9
MovieDB	16 MiB	1	19 MiB	868'980	6	7	8	0	4
KanjiDic2	13 MiB	1	18 MiB	917'833	3	27	10	0	6
XMark	11 MiB	1	13 MiB	324'274	2	74	9	0	13
Shakespeare	7711 KiB	1	9854 KiB	327'170	0	59	0	0	9
TreeOfLife	5425 KiB	1	7106 KiB	363'560	7	4	7	0	243
Thesaurus	4288 KiB	1	4088 KiB	201'798	7	33	9	0	7
MusicXML	3155 KiB	17	2942 KiB	171'400	8	179	56	0	8
BibDBPub	2292 KiB	3'465	2359 KiB	80'178	1	54	1	0	4
Factbook	1743 KiB	1	1560 KiB	77'315	16	23	32	0	6
XMark	1134 KiB	1	1334 KiB	33'056	2	74	9	0	13

Sources

Instances	Source
AirBase	[1]
Alfred	[2]
BibDBPub	[3]
CoPhIR	http://cophir.isti.cnr.it/
DBLP	http://dblp.uni-trier.de/xml
DBLP2	[3]
DDI	http://tools.ddialliance.org/
EnWikiMeta	[4]
EnWikipedia	[5]
EnWikiRDF	http://www.xml-benchmark.org/generated with xmlgen
EnWiktionary	[6]
EURLex	http://www.epsiplatform.eu/
Factbook	[7]
Freebase	[8]
FreeDB	[9]
Freshmeat	[10]
Genome1	[11]
HCIBIB2	[3]
Inex2009	[12]
IntAct	[13]
InterPro	[14]
iProClass	[15]
JMNEdict	[16]
KanjiDic2	[17]
MedLine	http://www.nlm.nih.gov/bsd
MeSH	[18]
MovieDB	[19]
MusicXML	[20]
Nasa	[7]
NewYorkTimes	[21]
OpenStreetMap	[22]
Organizations	http://www.data.gov/raw/1358
RuWikiHist	[23]
SDMX	[24]
Shakespeare	[25]
SwissProt	[26]
Thesaurus	[27]

Trebank	[28]
TreeOfLife	[29]
TrEMBL	[26]
Wikicorpus	[30]
XMark	http://www.xml-benchmark.org/ generated with xmlgen
ZDNET	[3]
ZhWikiHist	[31]
LibraryUKN	generated from university library data
MediaUKN	generated from university library data
DeepFS	generated from filesystem structure
University	generated from students test data
Feeds	compiled from news feeds
Twitter	compiled from Twitter feeds

References

- [1] <http://air-climate.eionet.europa.eu/databases/airbase/airbasexml>
- [2] <http://alfred.med.yale.edu/alfred/alfredWithDescription.zip>
- [3] <http://inex.is.informatik.uni-duisburg.de/2005/>
- [4] <http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-meta-current.xml.bz2>
- [5] <http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2>
- [6] <http://dumps.wikimedia.org/enwiktionary/latest/enwiktionary-latest-pages-meta-history.xml.7z>
- [7] <http://www.cs.washington.edu/research/xmldatasets/www/repository.html>
- [8] <http://download.freebase.com/wex>
- [9] <http://www.xml-databases.org/radio/xmlDatabases/projects/FreeDBtoXML>
- [10] <http://freshmeat.net/articles/freshmeat-xml-rpc-api-available>
- [11] ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/XML/ds_ch1.xml.gz
- [12] <http://www.mpi-inf.mpg.de/departments/d5/software/inex>
- [13] <ftp://ftp.ebi.ac.uk/pub/databases/intact/current/index.html>
- [14] ftp://ftp.bio.net/biomirror/interpro/match_complete.xml.gz
- [15] ftp://ftp.pir.georgetown.edu/pir_databases/iproclass/iproclass.xml.gz
- [16] ftp://ftp.monash.edu.au/pub/nihongo/enamdict_doc.html
- [17] <http://www.csse.monash.edu.au/~jwb/kanjdic2>
- [18] <http://www.nlm.nih.gov/mesh/xmlmesh.html>
- [19] <http://eagereyes.org/InfoVisContest2007Data.html>
- [20] <http://www.recordare.com/xml/samples.html>
- [21] <http://www.nytimes.com/ref/membercenter/nytarchive.html>
- [22] <http://dump.wiki.openstreetmap.org/osmwiki-latest-files.tar.gz>
- [23] <http://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-meta-history.xml.7z>
- [24] <http://www.metadatechnology.com/>
- [25] <http://www.cafeconleche.org/examples/shakespeare>
- [26] ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase
- [27] <http://www.drze.de/BELIT/thesaurus>
- [28] <http://www.cs.washington.edu/research/xmldatasets>
- [29] <http://tolweb.org/data/tolskeletaldump.xml>
- [30] <http://www-connex.lip6.fr/~denoyer/wikipediaXML>
- [31] <http://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-meta-history.xml.7z>

Storage Layout

This article is part of the Advanced User's Guide. It presents some low-level details on how data is stored in the database files.

Data Types

The following data types are used for specifying the storage layout:

Type	Description	Example (native → hex integers)
<code>Num</code>	Compressed integer (1-5 bytes), specified in <code>Num.java</code> ^[1]	15 → 0F; 511 → 41 FF
<code>Token</code>	Length (<code>Num</code>) and bytes of UTF8 byte representation	Hello → 05 48 65 6c 6c 6f
<code>Double</code>	Number, stored as token	123 → 03 31 32 33
<code>Boolean</code>	Boolean (1 byte, 00 or 01)	true → 01
<code>Nums</code> , <code>Tokens</code> , <code>Doubles</code>	Arrays of values, introduced with the number of entries	1,2 → 02 01 31 01 32
<code>TokenSet</code>	Key array (<code>Tokens</code>), next/bucket/size arrays (3x <code>Nums</code>)	

Database Files

The following tables illustrate the layout of the BaseX database files. All files are suffixed with `.baseX`.

Meta Data, Name/Path/Doc Indexes: `inf`

Description	Format	Method
1. Meta Data	1. Key/value pairs, in no particular order (<code>Token/Token</code>): <ul style="list-style-type: none"> • Examples: <code>FNAME</code>, <code>TIME</code>, <code>SIZE</code>, ... • <code>PERM</code> → Number of users (<code>Num</code>), and name/password/permission values for each user (<code>Token/Token/Num</code>) 2. Empty key as finalizer	<code>DiskData()</code> ^[2] <code>MetaData()</code> ^[3] <code>Users()</code> ^[4]
2. Main memory indexes	1. Key/value pairs, in no particular order (<code>Token/Token</code>): <ul style="list-style-type: none"> • <code>TAGS</code> → Tag Index • <code>ATTS</code> → Attribute Name Index • <code>PATH</code> → Path Index • <code>NS</code> → Namespaces • <code>DOCS</code> → Document Index 2. Empty key as finalizer	<code>DiskData()</code> ^[2]
2 a) Name Index Tag/attribute names	1. Token set, storing all names (<code>TokenSet</code>) 2. One <code>StatsKey</code> instance per entry: <ol style="list-style-type: none"> 2.1. Content kind (<code>Num</code>): <ol style="list-style-type: none"> 2.1.1. Number: min/max (<code>Doubles</code>) 2.1.2. Category: number of entries (<code>Num</code>), entries (<code>Tokens</code>) 2.2. Number of entries (<code>Num</code>) 2.3. Leaf flag (<code>Boolean</code>) 2.4. Maximum text length (<code>Double</code>; legacy, could be <code>Num</code>) 	<code>Names()</code> ^[5] <code>TokenSet.read()</code> ^[6] <code>StatsKey()</code> ^[7]

2 b) Path Index	<ol style="list-style-type: none"> 1. Flag for path definition (Boolean, always <code>true</code>; legacy) 2. PathNode: <ol style="list-style-type: none"> 2.1. Name reference (Num) 2.2. Node kind (Num) 2.3. Number of occurrences (Num) 2.4. Number of children (Num) 2.5. Double; legacy, can be reused or discarded 2.6. Recursive generation of child nodes ($\rightarrow 2$) 	PathSummary() [8] PathNode() [9]
2 c) Namespaces	<ol style="list-style-type: none"> 1. Token set, storing prefixes (TokenSet) 2. Token set, storing URIs (TokenSet) 3. NSNode: <ol style="list-style-type: none"> 3.1. pre value (Num) 3.2. References to prefix/URI pairs (Nums) 3.3. Number of children (Num) 3.4. Recursive generation of child nodes ($\rightarrow 3$) 	Namespaces() [10] NSNode() [11]
2 d) Document Index	Array of integers, representing the distances between all document pre values (Nums)	DocIndex() [12]

Node Table: **tbl**, **tbli**

- **tbl**: Main database table, stored in blocks.
- **tbli**: Database directory, organizing the database blocks.

Some more information on the Node table storage is available.

Texts: **txt**, **atv**

- **txt**: Heap file for text values (document names, string values of texts, comments and processing instructions)
- **atv**: Heap file for attribute values.

Value Indexes: **txtl**, **txtr**, **atvl**, **atvr**

Text Index:

- **txtl**: Heap file with ID lists.
- **txtr**: Index file with references to ID lists.

The **Attribute Index** is contained in the files **atvl** and **atvr**; it uses the same layout.

Full-Text Fuzzy Index: **ftxx**, **ftxy**, **ftxz**

...will soon be reimplemented.

Full-Text Trie Index: `ftxa`, `ftxb`, `ftxc`

...will soon be dismissed.

References

- [1] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/util/Num.java>
- [2] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/data/DiskData.java>
- [3] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/data/MetaData.java>
- [4] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/core/Users.java>
- [5] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/index/Names.java>
- [6] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/util/hash/TokenSet.java>
- [7] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/index/StatsKey.java>
- [8] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/index/path/PathSummary.java>
- [9] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/index/path/PathNode.java>
- [10] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/data/Namespace.java>
- [11] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/data/NSNode.java>
- [12] <https://github.com/BaseXdb/basex/blob/master/src/main/java/org/basex/index/DocIndex.java>

Node table storage

This article describes the Storage Layout of the main database table.

BaseX logically splits the `tbl.basex` file into blocks with length 4096 bytes, i.e. each block can have max 256 records each with length 16 bytes. The records within a block are sorted by their pre value (which, therefore, can be implicitly determined and need not be saved).

For each block BaseX stores in a separate file (`tbli.basex`) the smallest pre value within that block (and since the records are sorted, that will be the pre value of the first record stored in the block). These will be referred as `fpre` from now on. The physical address of each block is stored in `tbli.basex`, too.

Since these two maps will not grow excessively large, but are accessed resp. changed on each read resp. write operation, they are kept in main memory and flushed to disk on closing the database.

A newly created database with 256 + 10 records will occupy the first two blocks with physical addresses 0 and 4096. The corresponding `fpre`'s will be 0 and 256.

If a record with `pre = 12` is to be inserted, it needs to be stored in the first block, which is, however, full. In this case, a new block with physical address 8192 will be allocated, the records with `pre` values from 12 to 255 will be copied to the new block, the new record will be stored in the old block at `pre = 12`, and the two maps will look like this:

```
fpre's = 0, 13, 257  
addr's = 0, 8192, 4096
```

Basically, the old records remain in the first block, but they will not be read, since the `fpre`'s array says that only 13 records are stored in the first block. This causes redundant storage of the records with old `pre`s from 13 to 255.

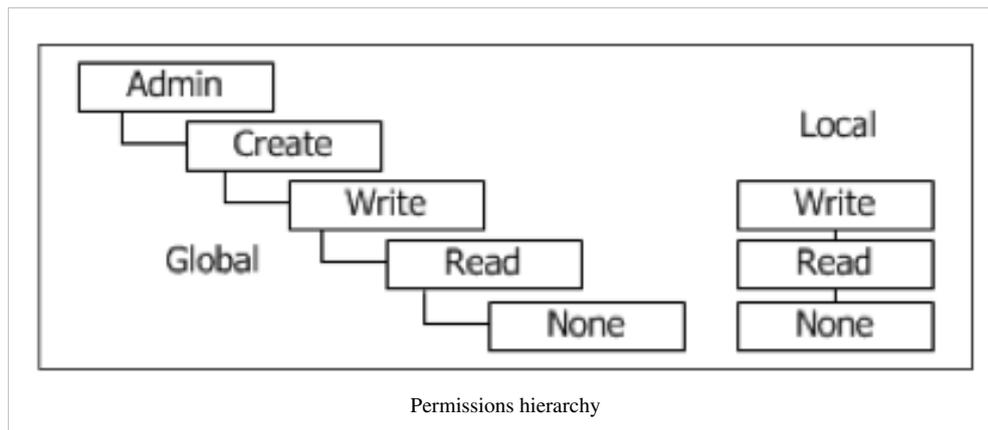
Additionally to these two maps (`fpre`'s and `addr`'s), BaseX maintains a bit map (which is also stored in `tbli.basex`) which reflects which physical blocks are free and which not, so that when a new block is needed, an already free one will be reused.

User Management

This article is part of the Advanced User's Guide. The user management defines which permissions are required by a user to perform a specific database command.

In the permission hierarchy below, the existing permissions are illustrated. A higher permission includes all lower permissions. For example, all users who have the `WRITE` permission assigned will also be able to execute commands requiring `READ` permission. Next, local permissions exist, which can be assigned to single databases. Local permission have a higher priority and override global permissions.

All global permissions are stored in the file `.basexperm`, and local permissions are encoded in the database meta data (`inf.basex`).



User names must follow the valid names constraints.

Commands

Admin permissions are needed to execute all of the following commands:

Creating user 'test' (password will be entered on command line):

```
> CREATE USER test
```

Change user 'test' password (password will be entered on command line):

```
> ALTER USER test
```

As global permissions, you can set 'none', 'read', 'write', 'create' and 'admin':

Grant all permissions to user 'test':

```
> GRANT admin TO test
```

Valid local permissions are 'none', 'read' and 'write':

Granting write permission on database 'factbook' to user 'test':

```
> GRANT write ON factbook TO test
```

Note: Local permissions overwrite global permissions. As a consequence, the 'test' user will only be allowed to access (i.e., read and write) the 'factbook' database. If no local permissions are set, the global rights are inherited.

Showing global permissions:

```
> SHOW USERS
```

Showing local permissions on database 'factbook':

```
> SHOW USERS ON factbook
```

Dropping of user 'test':

```
> DROP USER test
```

Transaction Management

This article is part of the Advanced User's Guide. The BaseX client-server architecture offers ACID safe transactions, with multiple readers and single writers. Here are some more informations about the transaction management.

Transaction

In a nutshell, a transaction is equal to a command or query. So each command or query sent to the server becomes a transaction.

Incoming requests are parsed and checked for errors on the server. If the command or query is not correct, the request will not be executed, and the user will receive an error message. Otherwise the request becomes a transaction and gets into the transaction monitor.

Note: An unexpected abort of the server during a transaction, caused by a hardware failure or power cut, will probably lead to an inconsistent database state if a transaction was active at the shutdown time. So we advise to use the BACKUP command to backup your database regularly. If the worst case occurs you can restore the database with the RESTORE command.

Update Transactions

Update transactions are mainly update queries. When executing a XQuery Update query, all update operations of the query are stored in a pending update list. They will be executed all at once, so the database is updated atomically. If any of the update sub-operations is erroneous, the overall transaction will be aborted.

Concurrency Control

The concurrency control checks for each transaction, which will perform a read or write operation on the database, the status of the lock object and decides whether the isolation is guaranteed for that transaction or not. If the isolation can be guaranteed the transaction will be started immediately. Otherwise, the transaction enters a waiting queue and waits till the transaction monitor validates and starts the transaction. The transaction monitor starts either the next writing transaction or the next group of reading transactions (if there are any on the queue).

Transaction Monitor

The transaction monitor ensures that just one writing transaction is active at the same time. This seems to be an odd mechanism, but it is needed since the complexity of updates increased and it is possible now to access multiple databases in one XQuery Update query.

To avoid starvation of any transaction and wrong execution orders the waiting queue works with the FIFO principle ('First-In First-Out'), which states that the first process that arrives at the server will be the first one that will be executed. The FIFO principle cannot be adhered in a group of reading transactions, as they run in different threads and thus can overtake each other.

The use of the monitor also prevents the system from deadlocks, because the critical resource is only assigned to one writing transaction resp. a group of reading transactions.

Locking

Update Operations

During the term of a database update, a locking file `upd.basex` will reside in that database directory. If the update fails for some unexpected reason, or if the process is killed ungracefully, this file may not be deleted. In this case, the database cannot be opened anymore using the default commands, and the message "Database ... is being updated, or update was not completed" will be shown instead. If the locking file is manually removed, you may be able to reopen the database, but you should be aware that database may have got corrupt due to the interrupted update process, and you should revert to the most recent database backup.

Database Locks

The standalone versions of BaseX (command-line, GUI) cannot be synchronized with other BaseX instances. If concurrent write operations are to be performed, we generally recommend users to always work with the client/server architecture.

Since [Version 7.2.1](#), to avoid database corruptions caused by unmanaged concurrent write operations, a shared lock is requested on the database table file (`tbl.basex`) whenever a database is opened. If an update operation is triggered, it will be rejected with the message "Database ... is opened by another process." if no exclusive lock can be acquired.

In [Version 7.2](#), pin files were written into the directory of opened databases. The message above appeared whenever an unused pin file remained after having ungracefully killed or shutting down a BaseX instance. In this cases, pin files could be manually removed.

Changelog

Version 7.2.1

- Updated: pin files replaced with shared/exclusive filesystem locking

Version 7.2

- Added: pin files to mark open databases

Version 7.1

- Added: update lock files
-

Logging

This article is part of the Advanced User's Guide. It describes how client operations are logged by the server.

Introduction

The server logs give you the following advantages:

- Overview about all processes executed on your server
- Tracing of any errors thrown by BaseX
- Tracing of wrong modified data
- Tracing of all specific user processes
- Tracing of time specific processes

The server logs are written in plain-text and can be opened with any editor. In your Database Directory, you can find a folder named `.logs` in which all log files are stored with the according date. Note that, depending on your OS and configuration, files and folders with a `'` prefix may be hidden.

Logformat

Example

```
16:00:00.094 [127.0.0.1:3920] LOGIN admin OK
16:00:00.104 [127.0.0.1:3920] XQUERY 1 to 10 OK 12.36 ms
16:00:00.114 [127.0.0.1:3920] LOGOUT admin OK
```

Now you can see the 'admin' user has the IP 127.0.0.1 with the socketport 3920. With this information you can identify each transaction of the corresponding user.

Example

```
10:06:36.498 [127.0.0.1:49990] LOGIN admin OK
10:06:53.297 [127.0.0.1:49990] XQUERY 1 to 10 OK      17.89 ms
10:07:03.353 [127.0.0.1:49993] LOGIN testuser OK
10:07:08.259 [127.0.0.1:49993] XQUERY 1 to z Error: Stopped at line 1, column 6: [XPDY0002] No context item set for 'z'. 9.69 ms
10:07:15.505 [127.0.0.1:49990] LOGOUT admin OK
10:07:19.790 [127.0.0.1:49993] LOGOUT testuser OK
```

Now you can see the 'admin' user has the IP 127.0.0.1 with the socketport 49990 and the 'testuser' has the IP 127.0.0.1 with the socketport 49993. With this information you can identify each transaction of the corresponding user. The 'testuser' for example executed a query which was aborted by a syntax error.

Events

This article is part of the Advanced User's Guide. It presents how to trigger database events and notify listening clients.

Introduction

The events feature enables users with admin permissions to create events, which can then be watched by other clients. All clients that have registered for an event will be notified if an event is triggered by another client.

Managing Events

CREATE EVENT [name]

Creates an event [name].

DROP EVENT [name]

Drops the event with the specified [name].

SHOW EVENTS

Shows all events.

Watching/Unwatching Events

The events can currently be watched by the Java ^[1] and C# ^[1] clients. See the following Java code example:

Watch events:

```
// name of the event
String event = "call";
// create new client
BaseXClient client = new BaseXClient("localhost", 1984, "admin", "admin");
// register for an event
client.watch(event, new EventNotifier() {
    @Override
    public void notify(final String value) {
        System.out.println("Received data: " + value);
    }
});
```

Unwatch events:

```
// unregister from an event
client.unwatch(event);
```

For a complete and self-contained example in Java, you may have a look [2].

Firing Events

Events are triggered via the XQuery function `db:event()`:

```
db:event($name as xs:string, $query as item())
```

Executes a `$query` and sends the resulting value to all clients watching the event with the specified `$name`. No event will be sent to the client that fired the event.

Example Scenarios

Basic

1. **Client1** creates an event with the name "EVENT"
2. **Client2** and **Client3** call the watch method for event "EVENT"
3. **Client1** executes XQuery

```
db:event("EVENT", "1 to 2")
```

4. **Client2** and **Client3** will receive the result 1 2
5. **Client2** executes XQuery

```
db:event("EVENT", "2 to 3")
```

6. **Client3** will receive the result 2 3

Included in Update Expression

1. **Client1** creates an event with the name "DELETED"
2. **Client2** and **Client3** call the watch method for event "DELETED"
3. **Client1** executes XQuery

```
let $deleted := //nodes return (  
  delete node $deleted,  
  db:event("DELETED", $deleted)  
)
```

4. **Client2** and **Client3** will receive the deleted nodes.

Included in Update Expression with Payload

1. **Client1** creates an event with the name "DELETED"
2. **Client2** and **Client3** call the watch method for event "DELETED"
3. **Client1** executes XQuery

```
let $deleted := //nodes return (  
  delete node $deleted,  
  db:event("DELETED",  
    <message>  
      <payload>{count($deleted)} items have been deleted.</payload>  
      <items>{$deleted}</items>  
    </message>  
)
```

4. **Client2** and **Client3** will receive the message with the payload and the deleted nodes.

References

- [1] <https://github.com/BaseXdb/basex-api/tree/master/src/main/java>
- [2] <https://github.com/BaseXdb/basex-examples/blob/master/src/main/java/org/basex/examples/api/EventExample.java>

Execution Plan

This article is part of the Advanced User's Guide. For each execution of a query, BaseX creates an execution plan. This execution plan shows you each step of the query, so that you can evaluate your query and analyse if it accesses any indexes or not. You can activate the execution plan by activating the `XMLPLAN` or `DOTPLAN` options.

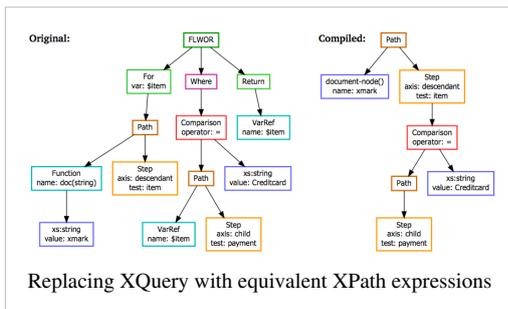
Examples

Execution plan for original and optimized query execution

Query: `for $item in doc('xmark')/descendant::item where $item/payment = 'Creditcard' return $item`

Optimized query: `doc('xmark')/descendant::item[payment = 'Creditcard']`

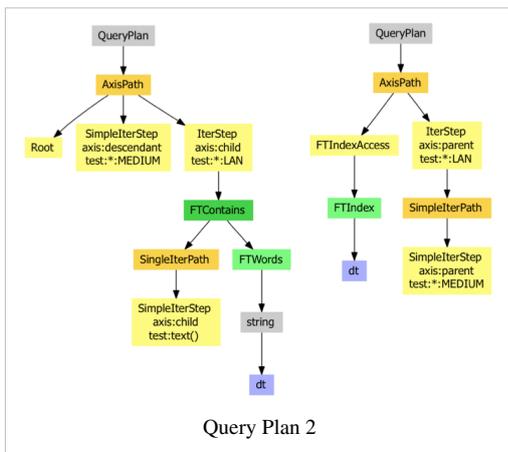
Execution plan:



Execution plan for query execution with full-text index access and without

Query: `//MEDIUM/LAN[text() contains text "dt"]`

Execution plan:



Article Sources and Contributors

Main Page *Source:* <http://docs.basex.org/index.php?oldid=6855> *Contributors:* AW, CG, Dimitar, Elmedin, Lukas.kircher, MediaWiki default, Michael, Ralf Jung, 44 anonymous edits

Getting Started *Source:* <http://docs.basex.org/index.php?oldid=6511> *Contributors:* AW, Arve, CG, Michael, 2 anonymous edits

Startup *Source:* <http://docs.basex.org/index.php?oldid=6724> *Contributors:* AW, Arve, CG, Dimitar, Lukas.kircher, 1 anonymous edits

Startup Options *Source:* <http://docs.basex.org/index.php?oldid=6081> *Contributors:* AW, CG, Holu, Michael

Start Scripts *Source:* <http://docs.basex.org/index.php?oldid=6743> *Contributors:* AW, CG, François Parmentier, Holu, Michael

Graphical User Interface *Source:* <http://docs.basex.org/index.php?oldid=6467> *Contributors:* AW, CG, Michael, 2 anonymous edits

Shortcuts *Source:* <http://docs.basex.org/index.php?oldid=5606> *Contributors:* CG, Jens Erat

Database Server *Source:* <http://docs.basex.org/index.php?oldid=6253> *Contributors:* AW, CG, Lukas.kircher

Standalone Mode *Source:* <http://docs.basex.org/index.php?oldid=6254> *Contributors:* AW, CG, Lukas.kircher

Web Application *Source:* <http://docs.basex.org/index.php?oldid=6380> *Contributors:* CG, Dimitar

Databases *Source:* <http://docs.basex.org/index.php?oldid=6806> *Contributors:* AW, CG, Jens Erat, Lukas.kircher, Michael

Binary Data *Source:* <http://docs.basex.org/index.php?oldid=5331> *Contributors:* CG

Parsers *Source:* <http://docs.basex.org/index.php?oldid=5995> *Contributors:* AW, CG, Jidanni, Lukas.kircher, Michael, Mroth, 1 anonymous edits

Commands *Source:* <http://docs.basex.org/index.php?oldid=6718> *Contributors:* AW, CG, Dimitar, Holu, Jan.vlcinsky, Jens Erat, Lukas.kircher, Michael

Options *Source:* <http://docs.basex.org/index.php?oldid=6844> *Contributors:* AW, CG, Holu, Lukas.kircher, Michael, Mroth

Integrating oXygen *Source:* <http://docs.basex.org/index.php?oldid=6772> *Contributors:* AW, CG, Michael

Integrating Eclipse *Source:* <http://docs.basex.org/index.php?oldid=6172> *Contributors:* AW, CG

XQuery *Source:* <http://docs.basex.org/index.php?oldid=6841> *Contributors:* CG

XQuery 3.0 *Source:* <http://docs.basex.org/index.php?oldid=6444> *Contributors:* CG, Dknippers, LeoWoerteler, Michael

Higher-Order Functions *Source:* <http://docs.basex.org/index.php?oldid=6033> *Contributors:* CG, LeoWoerteler, Michael

Module Library *Source:* <http://docs.basex.org/index.php?oldid=6567> *Contributors:* CG

Repository *Source:* <http://docs.basex.org/index.php?oldid=6843> *Contributors:* AW, CG, Lukas.kircher, Michael, Rosishadura

Java Bindings *Source:* <http://docs.basex.org/index.php?oldid=6842> *Contributors:* AW, CG, Dimitar, Dknippers, Michael, 7 anonymous edits

Full-Text *Source:* <http://docs.basex.org/index.php?oldid=6822> *Contributors:* CG, Dimitar, Michael, Piotr Banski, 8 anonymous edits

Full-Text: Japanese *Source:* <http://docs.basex.org/index.php?oldid=6395> *Contributors:* CG

XQuery Update *Source:* <http://docs.basex.org/index.php?oldid=6854> *Contributors:* AW, CG, Dknippers, Holu, Lukas.kircher, Michael, Mroth, 7 anonymous edits

Serialization *Source:* <http://docs.basex.org/index.php?oldid=6443> *Contributors:* CG, Jidanni, Michael

XQuery Errors *Source:* <http://docs.basex.org/index.php?oldid=6845> *Contributors:* CG, Lukas.kircher, Rosishadura

Cryptographic Module *Source:* <http://docs.basex.org/index.php?oldid=6569> *Contributors:* CG, Lukas.kircher, Michael

Database Module *Source:* <http://docs.basex.org/index.php?oldid=6817> *Contributors:* AW, CG, Dimitar, Holu, Lukas.kircher, Michael, 17 anonymous edits

File Module *Source:* <http://docs.basex.org/index.php?oldid=6707> *Contributors:* AW, CG, Dimitar, Michael, Rosishadura, 1 anonymous edits

Full-Text Module *Source:* <http://docs.basex.org/index.php?oldid=6734> *Contributors:* AW, CG

HTTP Module *Source:* <http://docs.basex.org/index.php?oldid=6573> *Contributors:* CG, Rosishadura

Higher-Order Functions Module *Source:* <http://docs.basex.org/index.php?oldid=6735> *Contributors:* CG, LeoWoerteler

Index Module *Source:* <http://docs.basex.org/index.php?oldid=6575> *Contributors:* AW, CG

JSON Module *Source:* <http://docs.basex.org/index.php?oldid=6759> *Contributors:* CG, Michael

Map Module *Source:* <http://docs.basex.org/index.php?oldid=6577> *Contributors:* CG, Dknippers, Holu, LeoWoerteler

Math Module *Source:* <http://docs.basex.org/index.php?oldid=6578> *Contributors:* AW, CG, Michael, 3 anonymous edits

Repository Module *Source:* <http://docs.basex.org/index.php?oldid=6846> *Contributors:* CG, Rosishadura

SQL Module *Source:* <http://docs.basex.org/index.php?oldid=6581> *Contributors:* CG, Michael, Rosishadura

Utility Module *Source:* <http://docs.basex.org/index.php?oldid=6825> *Contributors:* AW, CG, LeoWoerteler, Leonard.woerteler, Lukas.kircher, Michael, Mroth, 1 anonymous edits

XSLT Module *Source:* <http://docs.basex.org/index.php?oldid=6585> *Contributors:* CG

ZIP Module *Source:* <http://docs.basex.org/index.php?oldid=6586> *Contributors:* AW, CG

ZIP Module: Word Documents *Source:* <http://docs.basex.org/index.php?oldid=6214> *Contributors:* AW, CG

Developing *Source:* <http://docs.basex.org/index.php?oldid=6810> *Contributors:* AW, Arve, CG, Michael, Ralf Jung

Integrate *Source:* <http://docs.basex.org/index.php?oldid=6738> *Contributors:* AW, CG, Dimitar, Michael, Mroth

Git *Source:* <http://docs.basex.org/index.php?oldid=4622> *Contributors:* CG, Dimitar, Michael

Maven *Source:* <http://docs.basex.org/index.php?oldid=6824> *Contributors:* CG, Lukas.kircher, Michael, 2 anonymous edits

Releases *Source:* <http://docs.basex.org/index.php?oldid=6365> *Contributors:* CG

Translations *Source:* <http://docs.baseex.org/index.php?oldid=5625> *Contributors:* CG

REST *Source:* <http://docs.baseex.org/index.php?oldid=6823> *Contributors:* Arve, CG, Lukas.kircher, LukasL, Michael, Ralf Jung, Rosishadura

REST: POST Schema *Source:* <http://docs.baseex.org/index.php?oldid=6284> *Contributors:* CG

RESTXQ *Source:* <http://docs.baseex.org/index.php?oldid=6744> *Contributors:* Arve, CG, Michael

WebDAV *Source:* <http://docs.baseex.org/index.php?oldid=6390> *Contributors:* CG, Dimitar, Jens Erat

WebDAV: Windows 7 *Source:* <http://docs.baseex.org/index.php?oldid=6293> *Contributors:* CG

WebDAV: Windows XP *Source:* <http://docs.baseex.org/index.php?oldid=6291> *Contributors:* CG

WebDAV: Mac OSX *Source:* <http://docs.baseex.org/index.php?oldid=6292> *Contributors:* CG

WebDAV: GNOME *Source:* <http://docs.baseex.org/index.php?oldid=6290> *Contributors:* CG

WebDAV: KDE *Source:* <http://docs.baseex.org/index.php?oldid=6289> *Contributors:* CG

Clients *Source:* <http://docs.baseex.org/index.php?oldid=6807> *Contributors:* AW, CG, Holu, Michael

Standard Mode *Source:* <http://docs.baseex.org/index.php?oldid=5247> *Contributors:* AW, CG, Michael

Query Mode *Source:* <http://docs.baseex.org/index.php?oldid=6043> *Contributors:* AW, CG, Michael

PHP Example *Source:* <http://docs.baseex.org/index.php?oldid=6306> *Contributors:* AW, CG, Dimitar, Michael

Server Protocol *Source:* <http://docs.baseex.org/index.php?oldid=6410> *Contributors:* AW, CG, Lukas.kircher, Michael

Server Protocol: Types *Source:* <http://docs.baseex.org/index.php?oldid=6821> *Contributors:* CG

Java Examples *Source:* <http://docs.baseex.org/index.php?oldid=6815> *Contributors:* AW, CG, Lukas.kircher, Michael, Mroth, 1 anonymous edits

Advanced User's Guide *Source:* <http://docs.baseex.org/index.php?oldid=6659> *Contributors:* CG

Configuration *Source:* <http://docs.baseex.org/index.php?oldid=6277> *Contributors:* AW, CG, Lukas.kircher

Indexes *Source:* <http://docs.baseex.org/index.php?oldid=6853> *Contributors:* AW, CG, Lukas.kircher, Michael, Mroth, 4 anonymous edits

Backups *Source:* <http://docs.baseex.org/index.php?oldid=6189> *Contributors:* AW, CG, Dimitar, Michael

Catalog Resolver *Source:* <http://docs.baseex.org/index.php?oldid=5939> *Contributors:* CG, Gimsieke, Michael, Mroth, 3 anonymous edits

Statistics *Source:* <http://docs.baseex.org/index.php?oldid=6816> *Contributors:* AW, CG, Mroth

Storage Layout *Source:* <http://docs.baseex.org/index.php?oldid=6663> *Contributors:* CG

Node table storage *Source:* <http://docs.baseex.org/index.php?oldid=6666> *Contributors:* CG, Dimitar

User Management *Source:* <http://docs.baseex.org/index.php?oldid=6657> *Contributors:* AW, CG, Jens Erat, Kulnor, Michael

Transaction Management *Source:* <http://docs.baseex.org/index.php?oldid=6721> *Contributors:* AW, CG, Elmedin, Lukas.kircher, Michael, Mroth

Logging *Source:* <http://docs.baseex.org/index.php?oldid=6273> *Contributors:* AW, CG, Michael, Mroth

Events *Source:* <http://docs.baseex.org/index.php?oldid=6366> *Contributors:* AW, CG, Dimitar, Michael

Execution Plan *Source:* <http://docs.baseex.org/index.php?oldid=5409> *Contributors:* AW, CG, Dimitar, Leonard.woerteler, Michael

License

Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)
<http://creativecommons.org/licenses/by-sa/3.0/>
