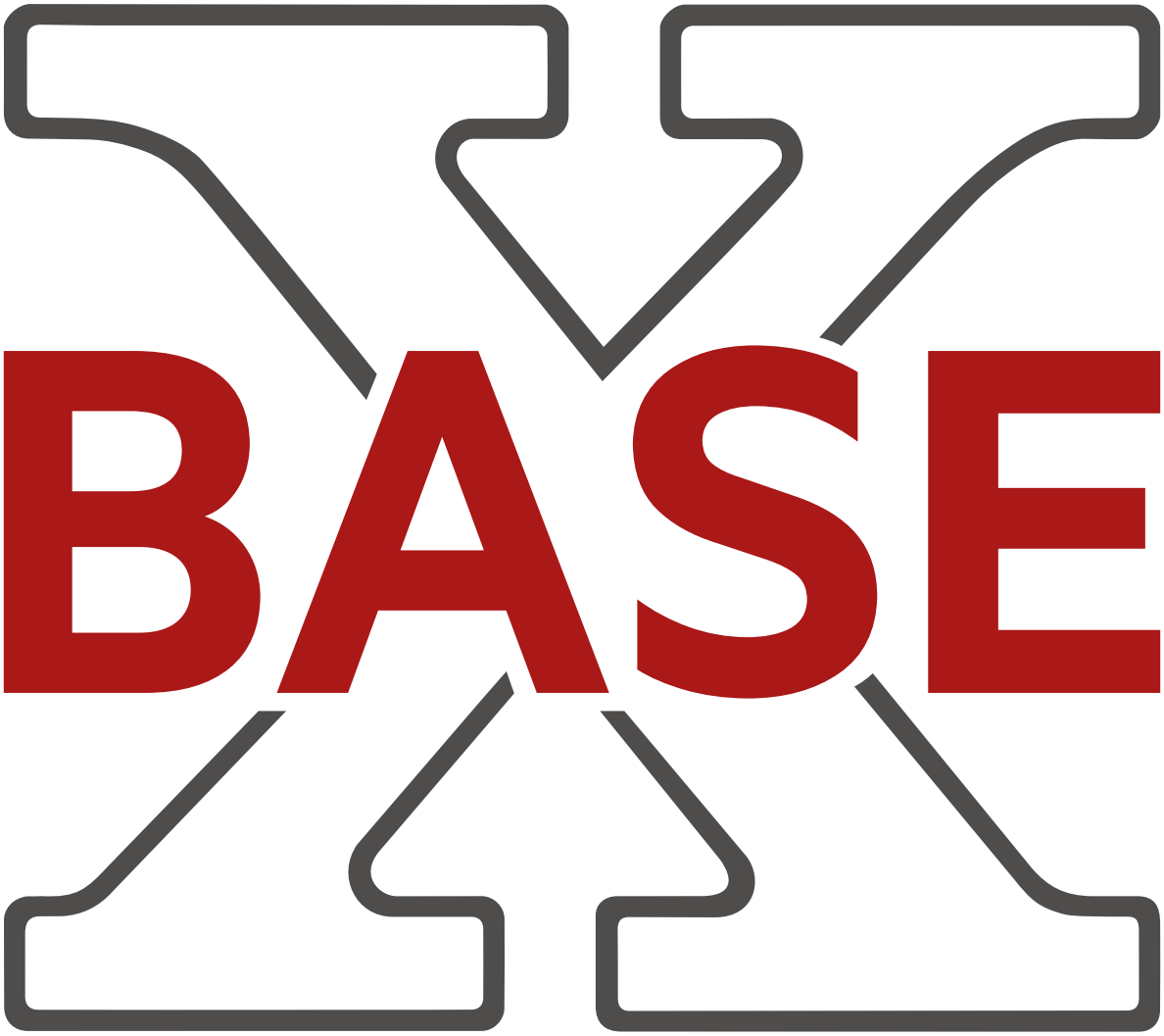
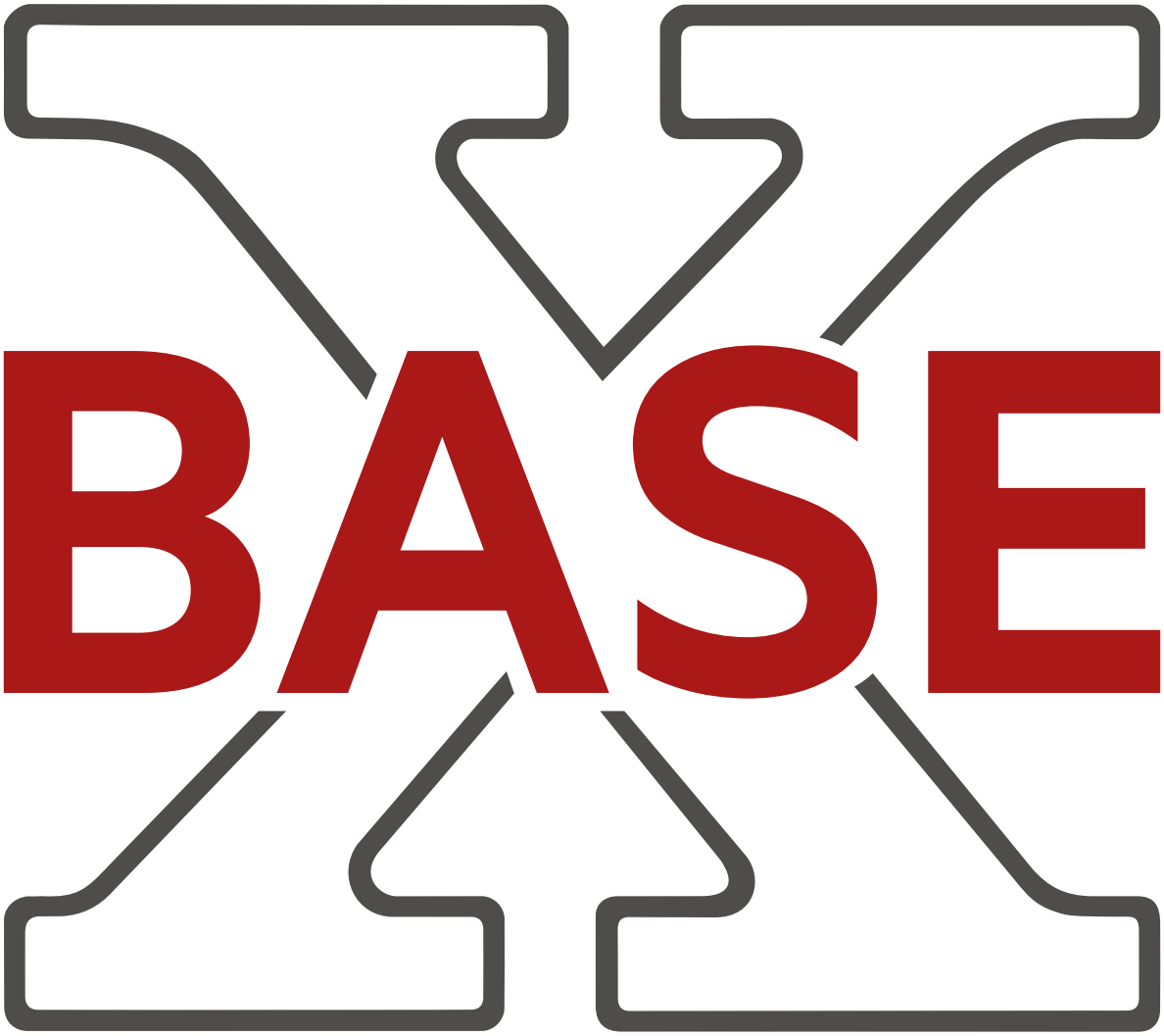
BaseX Documentation

Version 8.2



BaseX Documentation:

Version 8.2

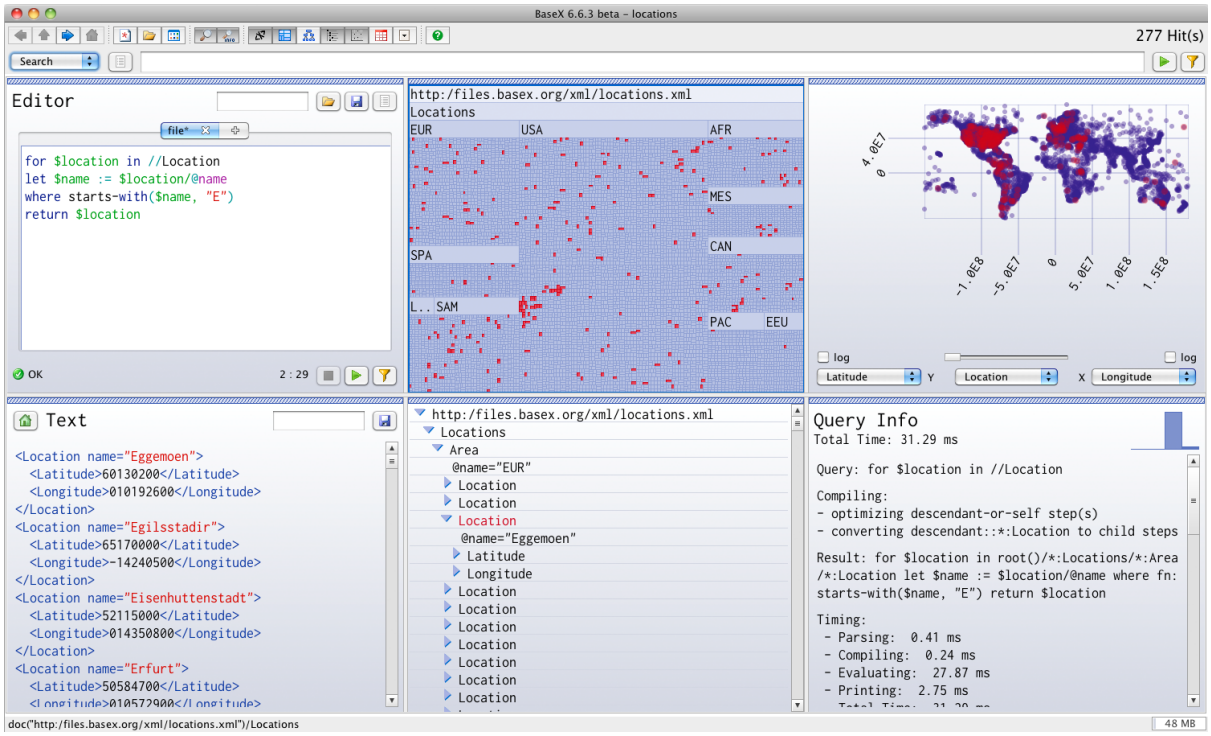


Publication date 2015-05-21

Table of Contents

Chapter 1. Main Page

Read this entry online in the BaseX Wiki.



BaseX GUI

Welcome to the documentation of BaseX!

BaseX is both a light-weight, high-performance and scalable XML Database and an XQuery 3.0 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).

This documentation is based on **BaseX 8.2**. It can also be downloaded as PDF. Features that have recently been added or changed are *highlighted* and can be searched for.

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 basex-talk mailing list.

Part I. Getting Started

Chapter 2. Getting Started

Read this entry online in the BaseX Wiki.

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

Overview

First Steps

- Startup : How to get BaseX running
- Command-Line 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
- DBA : Browser-based database administration

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

Tutorials and Slides

- BaseX for Dummies (I) , written by Paul Swennenhuis
- BaseX for Dummies (I), ZIP Package
- BaseX for Dummies (II)
- BaseX Adventures . Written by Neven Jovanovi#
- Tutorial . Written by Imed Bouchrika

- Slides and Examples from the BaseX User Meetups

- Stack Overflow , Questions tagged with `basex`
- XQuery Summer Institute , Exercises
- XQuery Tutorial , W3 Schools

Chapter 3. Startup

Read this entry online in the BaseX Wiki.

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

Getting Started

First of all, download BaseX from our homepage. The following distributions are available:

- the **Core Package** is a JAR file, which contains the database code, the query processor and the GUI frontend. It runs completely without additional libraries.
- the **ZIP Archive** and the **Windows Installer** contain libraries for web applications and advanced features, Start Scripts, and some additional optional files.
- the **WAR Archive** can be embedded in existing Java web servers.

Some additional distributions are available from the download page, most of which contain only the core package and, optionally, scripts for starting BaseX.

BaseX is very light-weight. It can be run and used in various ways:

- as standalone application, using the Standalone mode or the Graphical User Interface,
- as Client/Server application, or
- as Web Application, called from a web server.

It can also be embedded as a library in your own application.

Requirements

BaseX is platform-independent and runs on any system that provides an implementation of Java (JRE). Since **Version 8.0** of BaseX, Java 7 is mandatory, because it provides better file handling support, and because Oracle stopped public support for older versions.

BaseX has been tested on several platforms, including Windows (2000, XP, Vista, 7), Max OS X (10.x), Linux (SuSE xxx, Debian, Ubuntu) and OpenBSD (4.x).

Concurrent Operations

If you plan to perform concurrent read and write operations on a single database, you should use the client/server architecutre or deploy it as web application. You may safely open the same database in different JVMs (Java virtual machines) for read-only access, and you won't encounter any problems when reading from or writing to different databases, but your update operations will be rejected if the database to be written to is currently opened by another virtual machine.

Standalone

The Standalone Mode can be used to execute XQuery expressions or run database commands on command line. It can also be used both for scripting and batch processing your XML data. It 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.

It is important to remember that the standalone mode does *not* interact with the Client/Server architecture.

Graphical User Interface

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.

It 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.

Note that the GUI does *not* interact with the client/server architecture.

Client/Server

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.

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.

We provide additional clients in various programming languages.

HTTP Server

With the HTTP Server, BaseX can be used as Web Application. It provides access to the REST, RESTXQ and WebDAV services. An instance of the Jetty Web Server will be created, which by default listens to the port `8984`. Additionally, the BaseX Server will be started, which is accessible on port `1984`.

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

- Run one of the `basexhttp` or `basexhttp.bat` scripts. Call the script with the `stop` keyword to gracefully shut down the server.
- On *Windows*: Double click on the **BaseX Server** or **BaseX Server (stop)** icon.
- BaseX can also be deployed as web servlet.

After that, you can e. g. open your browser and navigate to the RESTXQ start page http://localhost:8984.

Changelog

Version 8.0

- Update: Switched to Java 7

Version 7.0

- Updated: BaseXJAXRX has been replaced with BaseXHTTP

Chapter 4. Command-Line Options

Read this entry online in the BaseX Wiki.

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

Options can be specified multiple times. All options are evaluated in the given order (in earlier versions, the sequential evaluation was limited to the specified inputs, query files, queries and commands, while all other options were initially set). The standard input can be parsed by specifying a single dash (`-`) as argument.

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 [-bcdiLoqrRsuvVwxXz] [input]
  [input]     Execute input file or expression
  -b<pars>    Bind external query variables
  -c<input>   Execute commands from file or string
  -d          Activate debugging mode
  -i<input>   Open XML file or database
  -I<input>   Assign input string to context
  -o<output>  Write output to file
  -q<expr>    Execute XQuery expression
  -r<num>     Set number of query executions
  -R          Turn query execution on/off
  -s<pars>    Set serialization parameter(s)
  -t[path]    Run tests in file or directory
  -u          Write updates back to original files
  -v/V        Show (all) process info
  -w          Preserve whitespaces from input files
  -x          Show query execution plan
  -X          Show query plan before/after compilation
  -z          Skip output of results
```

The meaning of all flags is listed in the following table. If an equivalent database option exists (which can be specified via the SET command), it is listed as well. For the examples to work escaping some characters might be necessary, depending on your Operating System.

Flag	Description	Option	Default	Examples
`[input]`	Evaluates the specified input: • The input string may point to an existing file. If the file suffix is `.bxs`, the file content will be evaluated as Command Script; otherwise, it will be evaluated as XQuery expression. • Otherwise, the input string itself is evaluated as XQuery expression.			• `"doc('X')//head"`• `query.xq`• `commands.bxs`

-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.	BINDINGS		• -bv=example "declare variable $v external; $v"• -b{URL}ln=value"declare namespace ns='URL'; declare variable $ns:ln external; $ns:ln"
-c<input>	Executes Commands: • Several commands in the input can be separated by semicolons (;). • If the specified input is a valid file reference or URL, its content will be executed instead. Empty lines and lines starting with the number sign # will be ignored.			• -c"list;info"• -ccommands.txt• -c"<info/>"
-d	Toggles the debugging mode. Debugging information is output to *standard error*.	DEBUG	false	
-i<input>	Opens an XML file or database specified by the argument. The opened input can then be processed by a command or XQuery expression.			-iitems.xml "//item"
-I<input>	Assigns an input string as item of type xs:untypedAtomic to the query context.			-I "Hello Universe" -q "."
-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"
-r<num>	Specifies how often a specified query will be evaluated.	RUNS	1	-V -r10 "1"
-R	Specifies if a query will be executed or parsed only.	RUNQUERY	true	-V -R "1"
-s<pars>	Specifies parameters for serializing XQuery results; see Serialization for more details. This flag may be specified multiple times. Key and values are separated by the equality sign (=).	SERIALIZER		-smethod=text
-t	Runs all Unit tests in the specified file or directory.			-t project/tests
-u	Propagates updates on input files back to disk.	WRITEBACK	false	
-v	Prints process and timing information to the *standard output*.		false	

-V	Prints detailed query information to the *standard output*, including details on the compilation and profiling steps.	QUERYINFO	false	
-w	Specifies if whitespaces in XML text nodes should be chopped (which is the default) or preserved.	CHOP	true	
-x	This flags turn on the output of the query execution plan, formatted in XML.	XMLPLAN	false	
-X	Generates the query plan before or after query compilation. -x needs to be activated to make the plan visible.	COMPPLAN	true	
-z	Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.	SERIALIZE	true	

Server

Launch the server

```
$ basexserver
BaseX [Server]
Server was started (port: 1984)
```

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

```
$ basexserver -h
BaseX [Server]
Usage: basexserver [-cdinpSz] [stop]
  stop      Stop running server
  -c<cmds>  Execute initial database commands
  -d        Activate debugging mode
  -n<name>  Set host the server is bound to
  -p<port>  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). For the examples to work escaping some characters might be necessary, depending on your Operating System.

Flag	Description	Option	Default	Examples
stop	Stops an existing server instance and quits.			
-c<cmd>	Launches database commands before the server itself is started. Several commands can be separated by semicolons (;).			-c"open database;info"
-d	Turns on the debugging mode. Debugging information is output to *standard error*.	DEBUG	false	
-n<name>	Specifies the host the server will be bound to.	SERVERHOST		-p127.0.0.1
-p<num>	Specifies the port on which the server will be addressable.	SERVERPORT	1984	-p9999
-S	Starts the server as service (i.e., in the background).			
-z	Does not generate any log files.	LOG	true	

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.

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 [-bcdiLnopPqrRsUvVwxXz] [input]
  [input]     Execute input file or expression
  -b<pars>    Bind external query variables
  -c<input>   Execute commands from file or string
  -d          Activate debugging mode
  -i<input>   Open XML file or database
  -I<input>   Assign input string to context
  -n<name>    Set server (host) name
  -o<output>  Write output to file
  -p<port>    Set server port
  -P<pass>    Specify user password
  -q<expr>    Execute XQuery expression
  -r<num>     Set number of query executions
  -R          Turn query execution on/off
  -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
  -X          Show query plan before/after compilation
  -z          Skip output of results
```

The flags have the following meaning (equivalent database options are shown in the table as well). For the examples to work escaping some characters might be necessary, depending on your Operating System.

Flag	Description	Option	Default	Examples
`[input]`	Evaluates the specified input: • The input string may point to an existing file. If the file suffix is `.bxs`, the file content will be evaluated as Command Script; otherwise, it will be evaluated as XQuery expression. • Otherwise, the input string itself is evaluated as XQuery expression.			• `"doc('X')//head"`• `query.xq`• `commands.bxs`
`-b<pars>`	Binds external variables to XQuery expressions. This flag may be specified multiple times. Variables names and their	`BINDINGS`		• `-b $v=example "declare`

	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 or Expanded QName Notation.			`variable $v external; $v"• -b{URL}ln=value"declare namespace ns='URL'; declare variable $ns:ln external; $ns:ln"`
`-c<input>`	Executes Commands: • Several commands in the input can be separated by semicolons (`;`). • If the specified input is a valid file reference or URL, its content will be executed instead. Empty lines and lines starting with the number sign # will be ignored.			`• -c"list;info"• -ccommands.txt• -c"<info/>"`
`-d`	Toggles the debugging mode. Debugging information is output to *standard error*.	`DEBUG`	`false`	
`-i<input>`	Opens an XML file or database specified by the argument. The opened input can then be processed by a command or XQuery expression.			`-iitems.xml "//item"`
`-I<input>`	Assigns an input string as item of type `xs:untypedAtomic` to the query context.			`-I "Hello Universe" -q "."`
`-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. *Warning*: when the password is specified with this flag, it may get visible to others.	`PASSWORD`		`-Uadmin -Padmin`
`-q<expr>`	Executes the specified string as XQuery expression.		`-q"doc('input')//head"`	
`-r<num>`	Specifies how often a specified query will be evaluated.	`RUNS`	`1`	`-V -r10 "1"`
`-R`	Specifies if a query will be executed or parsed only.	`RUNQUERY`	`true`	`-V -R "1"`
`-s<pars>`	Specifies parameters for serializing XQuery results; see Serialization 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.	USER		-Uadmin
-v	Prints process and timing information to the *standard output*.		false	
-V	Prints detailed query information to the *standard output*, including details on the compilation and profiling steps.	QUERYINFO	false	
-w	Specifies if whitespaces in XML text nodes should be chopped (which is the default) or preserved.	CHOP	chop	
-x	This flags turn on the output of the query execution plan, formatted in XML.	XMLPLAN	false	
-X	Generates the query plan before or after query compilation. -x needs to be activated to make the plan visible.	COMPPLAN	after	
-z	Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.	SERIALIZE	true	

HTTP Server

Launch the HTTP server

```
$ basexhttp
BaseX [Server]
Server was started (port: 1984)
HTTP Server was started (port: 8984)
```

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

```
$ basexhttp -h
BaseX [HTTP]
Usage: basexhttp [-dhlnpPRUWz] [stop]
  stop      Stop running server
  -d        Activate debugging mode
  -h<port>  Set port of HTTP server
  -l        Start in local mode
  -n<name>  Set host name of database server
  -p<port>  Set port of database server
  -P<pass>  Specify user password
  -s<port>  Specify port to stop HTTP server
  -S        Start as service
  -U<name>  Specify user name
  -z        Suppress logging
```

The flags have the following meaning (equivalent database options are shown in the table as well). For the examples to work escaping some characters might be necessary, depending on your Operating System.

Flag	Description	Option	Default	Examples
stop	Stops a running HTTP server. By default, the database server will be stopped as well, unless -l has been specified.	pom.xml		
-d	Turns on the debugging mode. Debugging information is output to *standard error*.	DEBUG		

-h<num>	Specifies the port on which the HTTP server will be addressable.	jetty.xml	8984	-h9999
-l	Starts the server in *local mode*, and executes all commands in the embedded database context.	HTTPLOCAL		
-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 database server will be addressable.	SERVERPORT	1984	-p9998
-P<pass>	Specifies a user password, which will be used by the HTTP services to open a new session. If this flag is omitted, and if -U was specified, the password will be requested on command line. *Warning*: when the password is specified with this flag, it may get visible to others.	PASSWORD		-Uadmin -Padmin
-s<num>	Specifies the port that will be used to stop the HTTP server.	STOPPORT orpom.xml	8985	
-S	Starts the server as service (i.e., in the background).			
-U<name>	Specifies a user name, which will be used by the HTTP services for opening a new session.	USER		-Uadmin
-z	Does not generate any log files.	LOG		

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.

Changelog

Version 8.2

- Removed: Event ports, -e.

Version 8.1

- Added: Bind input strings to the query context with -I.

Version 8.0

- Removed: Command-line option -L (results will now be automatically separated by newlines).

Version 7.9

- Added: Runs tests in file or directory with -t.
- Removed: interactive server mode.

Version 7.8

- Added: Specify if a query will be executed or parsed only with `-R`.

Version 7.7

- Added: Bind host to the BaseX Server with `-n`.

Version 7.5

- Added: detection of Command Scripts.
- Removed: HTTP server flags `-R`, `-W`, and `-X`.

Version 7.3

- Updated: all options are now evaluated in the given order.
- Updated: Create main-memory representations for specified sources with `-i`.
- Updated: Options `-C`/`-c` and `-q`/`[input]` merged.
- Updated: Option `-L` also separates serialized items with newlines (instead of spaces).

Version 7.2

- Added: RESTXQ Service

Version 7.1.1

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

Version 7.1

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

Chapter 5. Start Scripts

Read this entry online in the BaseX Wiki.

The following scripts, which are referenced in the Startup and Command-Line Options articles, are also included in the Windows and ZIP distributions.

- We recommend you to manually add the `bin` directory of your BaseX directory to the PATH variable of your environment.
- The Windows installer automatically adds the project's `bin` directory to your path environment.
- If you work with Maven, you can directly run the scripts in the basex-core/etc and basex-api/etc sub-directories of the project.

If BaseX terminates with an `Out of Memory` error, you can assign more RAM via the `-Xmx` flag (see below).

Main Package

The following scripts can be used to 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

```
@echo off
setLocal EnableDelayedExpansion

REM Path to this script
set PWD=%~dp0

REM Core and library classes
set CP=%PWD%/../BaseX.jar
set LIB=%PWD%/../lib
for /R "%LIB%" %%a in (*.jar) do set CP=!CP!;%%a

REM Options for virtual machine
set VM=-Xmx512m

REM Run code
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 )"

# 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

# Run code
java -cp "$CP" $VM org.basex.BaseX "$@"
```

HTTP Server

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

Windows: basexhttp.bat

```
@echo off
setLocal EnableDelayedExpansion

REM Path to this script
set PWD=%~dp0

REM Core and library classes
set CP=%PWD%/../BaseX.jar
set LIB=%PWD%/../lib
for /R "%LIB%" %%a in (*.jar) do set CP=!CP!;%%a
for /R "%LIB%" %%a in (*.jar) do set CP=!CP!;%%a

REM Options for virtual machine
set VM=-Xmx512m

REM Run code
java -cp "%CP%;." %VM% org.basex.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 )"
BXCORE="$( cd -P "$BX/../basex" && pwd )"

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

# Options for virtual machine
VM=-Xmx512m

# Run code
java -cp "$CP" $VM org.basex.BaseXHTTP "$@"
```

Changelog

Version 7.5

- Updated: Static dependencies removed from Windows batch scripts.

Version 7.2

- Updated: The `BaseXHTTP` start class moved from `org.basex.api` to `org.basex`.

Version 7.0

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

Part II. User Interfaces

Chapter 6. Graphical User Interface

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. The BaseX homepage gives you a visual impression of the graphical user interface (GUI) of BaseX, and an introductory video is available, which presents some of its interactive features.

Startup

First of all, 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 (1.3 MB). If you type nothing in the input field, an empty database will be created. Next, choose the *OK* button, and BaseX will create a database that you can visually explore and query.

If no XML document is available, the Text Editor can also be used to create an initial XML document. After saving the entered XML document to harddisk, it can be specified in the above dialog.

Realtime Options

Via the *Options* menu, you can change 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` andd `/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
```

Text Editor

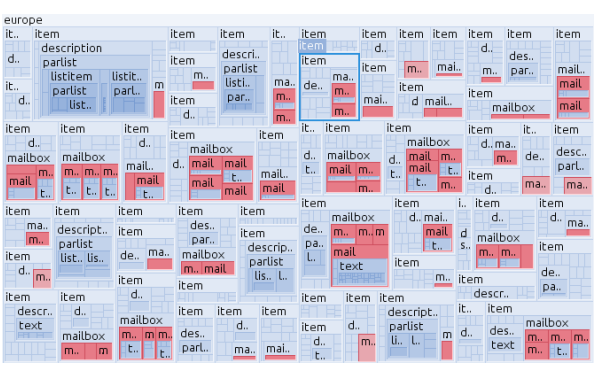
The text editor can be used to type in XQuery expressions, Command Scripts, XML documents, or any other text files. Query files and XML documents can be started by clicking on the green triangle. They will automatically parsed with each key click, and errors will be highlighted. Various keyboard shortcuts are available to speed up editing and debugging.

Visualizations

The BaseX GUI offers various visualizations, which help you to explore your XML data instances from different perspectives:

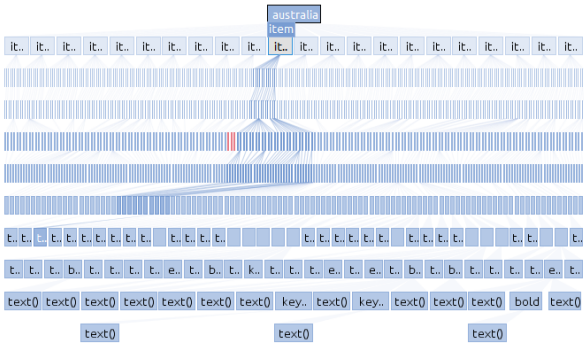
Text View **Result**

Displays query results and other textual output. Query results can be saved in a file.



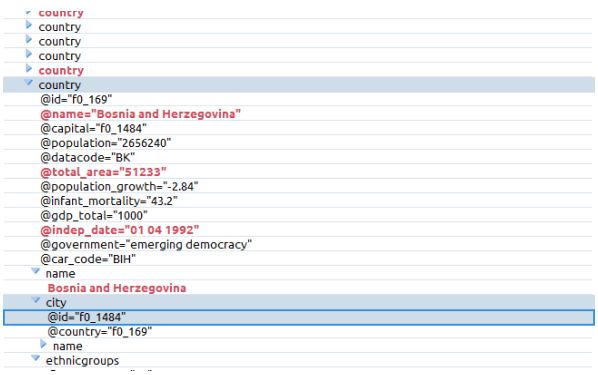
Map View **Map**

Displays all data in a TreeMap. All nodes of the XML document are represented as rectangles, filling the complete area. You can choose different layout algorithms in the Menu *Options* → *Map Layout*.



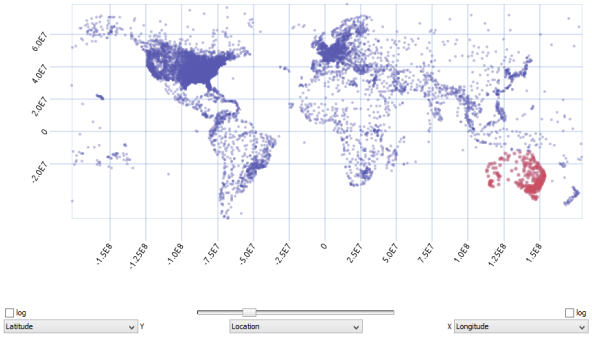
Tree View **Tree**

Displays all XML nodes in a top down tree with edges and nodes. You can change some settings of the Tree in the Menu *Options* → *Tree Options*.



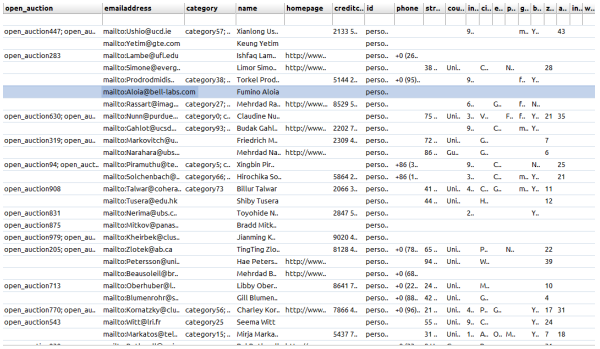
Folder View **Folder**

Displays all nodes in an Explorer-like folder view. Nodes can be expanded or closed by clicking on the arrows.



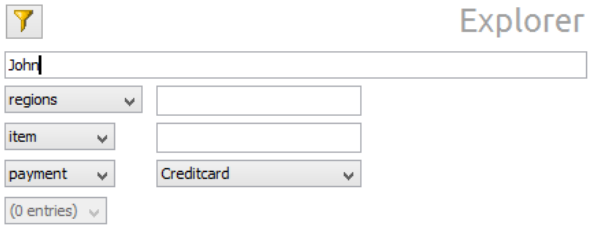
Scatterplot View **Plot**

Displays all nodes in a scatterplot, which is particularly helpful if you want to explore analyze your data. Three drop down menus allow custom axis assignments.



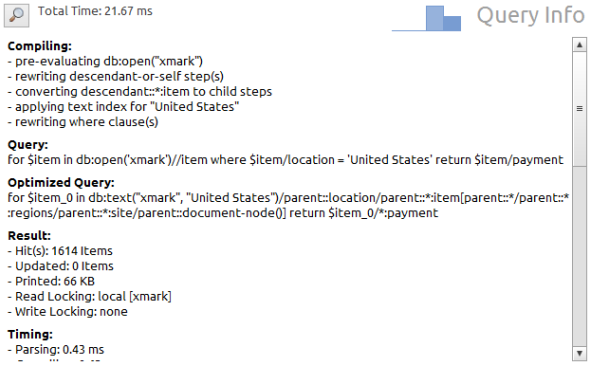
The Table View **Table**

Comes in handy if your data is highly regular. It displays all nodes in a table with rows and columns. Different assignments can be chosen by clicking on the arrow in the right upper corner.



Explorer View **Explorer**

Can be used to explore the contents of your database via drop-down menus, search fields and double sliders.



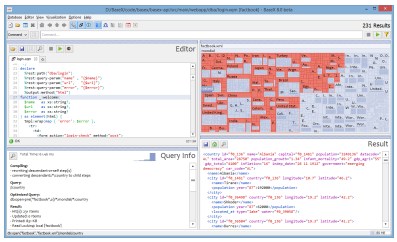
Info View **Info**

Helpful for analyzing the query plans of your XQuery expressions. It also displays information on the compilation and evaluation of queries.

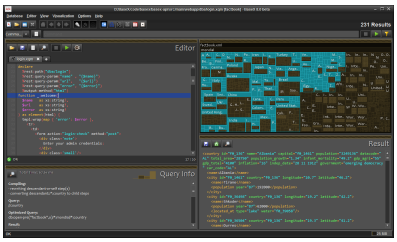
Look and Feels

By default, the Look and Feel of your operating system will be used in the GUI. In the *Preferences* dialog, you can choose among some more window themes.

The JTattoo library offers some more look and feels. If you download and copy the JTattoo jar file into the `lib` directory provided by the ZIP and EXE distribution of BaseX, 13 additional looks and feels will get available.



Default Look & Feel



HiFi Look & Feel

Changelog

Version 8.0

- Updated: support for dark look and feels; support for JTatto library

Chapter 7. Shortcuts

Read this entry online in the BaseX Wiki.

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

Editor

Code Completions

The GUI editor provides various code completions, which simplify the authoring of complex XQuery applications. Opening elements, comments, quotes or brackets will automatically be closed, and new lines will automatically be indented.

If some characters have been entered, and if the shortcut for code completions is pressed (Ctrl Space), a popup menu will appear and provides some code templates. If only one completion is possible, it will automatically be inserted.

Editor Shortcuts

The text editor can be used to create, edit, save and execute XQuery expressions, XML documents and any other textual files.

Custom Editing

Description	Win/Linux	Mac
Performs Code Completions	Ctrl Space	Ctrl Space
Sort lines	Ctrl U	# U
Format code (experimental)	Ctrl Shift F	# Shift F
(Un)comment selection/line	Ctrl K	# K
Delete complete line	Ctrl Shift D	# Shift D

Standard Editing

Description	Win/Linux	Mac
Undo recent changes	Ctrl Z	# Z
Redo recent changes	Ctrl Y	# Shift Z
Cut selection	Ctrl XCtrl Delete	# X
Copy selection to clipboard	Ctrl CCtrl Insert	# C
Paste from clipboard	Ctrl VShift 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

Finding

Description	Win/Linux	Mac
Jump to next error	Ctrl . (period)	# . (period)
Jump to currently edited file	Ctrl J	# J
Go to line	Ctrl L	# L
Find and replace text	Ctrl F	# F
Find next instance of text	F3Ctrl G	# F3# G
Find previous instance of text	Shift F3Ctrl Shift G	# Shift F3# Shift G

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 ↑/↓

GUI

Global Shortcuts

The following shortcuts are available from most GUI components:

Description	Win/Linux	Mac
Jump to input bar	F6	# 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

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 M	# M
View/edit database properties	Ctrl D	# D
Close opened database	Ctrl Shift W	# Shift W

Exit application	Ctrl Q	# Q

Editor

Description	Win/Linux	Mac
Create new tab	Ctrl T	# T
Open existing file	Ctrl O	# O
Save file	Ctrl S	# S
Save copy of file	Ctrl Shift S	# Shift S
Close tab	Ctrl W, Ctrl F4	# W, # F4

View

Description	Win/Linux	Mac
Toggle query/text editor	Ctrl E	# E
Toggle project structure	Ctrl P	# P
Find files	Ctrl H	# Shift H
Toggle result view	Ctrl R	# R
Toggle query info view	Ctrl I	# I

Options

Description	Win/Linux	Mac
Open preference dialog	Ctrl Shift P	# , (comma)

Visualization

Description	Win/Linux	Mac
Toggle map view	Ctrl 1	# 1
Toggle tree view	Ctrl 2	# 2
Toggle folder view	Ctrl 3	# 3
Toggle plot view	Ctrl 4	# 4
Toggle table view	Ctrl 5	# 5
Toggle explorer view	Ctrl 6	# 6

Help

Description	Win/Linux	Mac
Show Help	F1	F1

Additionally, the names of HTML entities will be converted to their Unicode representation (as an example, `Auml` will be translated to `ä`).

Changelog

Version 8.0

- Added: New code completions, popup menu

Version 7.8.2

- Added: Sort lines (Ctrl-U)

Version 7.8

- Added: Code Completions, Project (Ctrl P), Find Files (Ctrl Shift F)

Version 7.5

- Added: go to line (Ctrl F)

Version 7.3

- Added: delete complete line (Ctrl Shift D), jump to highlighted error (Ctrl .)

Chapter 8. Database Server

Read this entry online in the BaseX Wiki.

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, 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.
- Save this document to the directory you are working in.
- In the client terminal, type in:

```
> CREATE 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.

```
> XQUERY //country
```

- You can also run queries in files:

```
> RUN /Users/John/query.xq
```

Create a new database

Now we will create another database from the xmark.xml document.

- Create the new database, named 'xmark'.

```
> CREATE DB xmark xmark.xml
```

- Set the new database xmark as the context:

```
> OPEN xmark
```

- Now you can easily execute queries on your new database:

```
> XQUERY //people/person/name
```

Switch the database

- You can explicitly query the factbook database with the `doc(...)` funtion, no matter what the current context is.

```
> XQUERY doc("factbook")//country
```

- Otherwise, to set factbook as the current context, execute the following:

```
> OPEN factbook
```

- The following command lists all databases than can be opened by the currently logged in users:

```
> LIST
```

Close or delete a database

- To close the current context database, please type:

```
> CLOSE
```

- Use the DROP DB command to delete the xmark database:

```
> DROP 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:

```
> OPEN factbook
```

- Now add the xmark.xml document:

```
> ADD xmark.xml
```

Delete a document

- Deleting a document from a collection is easy:

```
> DELETE 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:

```
> DROP 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:

```
> LIST
```

- To see the general information of the opened database, type:

```
> INFO
```

- To list all sessions that are managed by the server instance, type:

```
> SHOW 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.

Chapter 9. Standalone Mode

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. BaseX offers a standalone 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 (note that the GUI does *not* interact with the client/server architecture).

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 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.

Chapter 10. Web Application

Read this entry online in the BaseX Wiki.

BaseX can be used as **Web Application**. The following three HTTP services are available:

- RESTXQ allows you to write web applications with XQuery,
- REST offers a RESTful API for accessing database resources via URLs, and
- WebDAV provides access to databases via the file system.

This article describes different ways of deploying and configuring these services. The services can be deployed as follows:

- as standalone application by running the BaseX HTTP Server,
- as web servlet in a Servlet Container, and
- as web servlet, using Maven.

Servlet Container

In order to deploy BaseX HTTP Services in a servlet container, you may download the WAR distribution of BaseX from the download site or compile it via `mvn compile war:war` in the `basex-api` package. The WAR file can then be deployed following the instructions of the corresponding servlet container (jetty, tomcat).

Configuring port, context path, etc. can be done by 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).

If run on a Jetty server you may use a `jetty.xml` file for detailed server configuration. You can e.g. enable SSL connections or Jetty logging. Place the `jetty.xml` right next to the `web.xml`. For detailed configuration refer to the Jetty Documentation. A sample jetty.xml is placed in the basex-api package.

To run on Apache Tomcat, start the tomcat server and add any `*.war` distribution to deploy using the Tomcat web interface. By default, the interface is accessible via http://localhost:8080/manager/html/.

Configuration

All database options can be specified in the `web.xml` file by prefixing the key with `org.basex.`. The most important options for the web application context are as follows:

Option	Default	Description
USER	`admin`	Applies to REST and WebDAV service: If no user is specified, the credentials must be passed on by the client. Please check by yourself if it is safe to store your credentials in plain text.
PASSWORD	`admin`	Applies to REST and WebDAV service: If no password is specified, it must be passed on by the client. Please check by yourself if it is safe to store your credentials in plain text.
HTTPLOCAL	`false`	Operation mode. By default, a database server instance will be started, as soon as the first HTTP service is called. The database server can be disabled by setting this flag to `true`.
RESTXQPATH	`.`	Relative or absolute directory referencing the RESTXQ modules. By default, the option points to the standard web application directory.
RESTPATH	`.`	Relative or absolute directory referencing queries and command-scripts that can be invoked via the run operation of REST. By default, the option points to the standard web application directory.

AUTHMETHOD	`Basic`	The default authentication method proposed by the server. The available methods are `Basic` and `Digest`.

Path options may contain an absolute or relative path. If a relative path is specified, its root will be the servlet (`webapp`) path:

```
<context-param>
  <param-name>org.basex.dbpath</param-name>
  <!-- will be rewritten to ..../webapp/WEB-INF/data -->
  <param-value>WEB-INF/data</param-value>
</context-param>
<context-param>
  <param-name>org.basex.repopath</param-name>
  <!-- will be kept as is -->
  <param-value>f:/basex/repository</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 by overriding the web.xml file. Another option is to directly edit the `WEB-INF/web.xml` file in the WAR archive (WAR files are simple ZIP files). Refer to the sample web.xml of the basex-api package.

Different credentials can be assigned to the REST and WebDAV service by specifying local init parameters. In the following example, specific credentials are set for the REST service:

```
<servlet>
  <servlet-name>REST</servlet-name>
  <servlet-class>org.basex.http.rest.RESTServlet</servlet-class>
  <init-param>
    <param-name>org.basex.user</param-name>
    <param-value>rest-user</param-value>
  </init-param>
  <init-param>
    <param-name>org.basex.password</param-name>
    <param-value>(:87!7X3$o3p</param-value>
  </init-param>
</servlet>
```

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
Default web server	`http://[host]:[port]/[servlet_context_path]/static`Before: `http://[host]:[port]/[servlet_context_path]`	Access your standard web files (e.g. HTML, JavaScript or CSS).
RESTXQ	`http://[host]:[port]/[servlet_context_path]`Before: `http://[host]:[port]/[servlet_context_path]/restxq`	Create XQuery web services and applications.
REST	`http://[host]:[port]/[servlet_context_path]/rest`	Access XML database and its resources.

WebDAV	`http://[host]:[port]/[servlet_context_path]/webdav` or `webdav://[host]:[port]/[servlet_context_path]/webdav` (depending on client)	Access databases via the filesystem.

Maven

Checkout the BaseX sources via Eclipse or Git. Execute `mvn install` in the `basex-core` 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 looks up all files in the directory `basex-api/src/main/webapp`. Jetty and servlet options can be configured in the `jetty.xml` and `web.xml` files as described above in the Servlet Container Configuration. The Jetty stop port can be changed in the Maven Jetty Plugin sesion in the `pom.xml` file.

User Management

By default, the REST and WebDAV services require client-side authentication. Default credentials can be stored server-side in the `web.xml` file or specified via command-line arguments. If the HTTP server is started with no pre-defined credentials, users and passwords can be sent via HTTP Basic Authentication or Digest Authentication.

Users are specified in a `users.xml` file, which is stored in the database directory (see User Management for more information).

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/
```

Changelog

Version 8.0

- Added: digest authentication
- Updated: user management
- Updated: default user/password disabled in web.xml

Version 7.7

- Added: service-specific permissions

Version 7.5

- Added: `jetty.xml`: configuration for Jetty Server
- Updated: `server` replaced with `httplocal` mode

Version 7.3

- Updated: `client` mode replaced with `server` mode

Version 7.2

- Web Application concept revised

Chapter 11. DBA

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section.

The full distributions of BaseX are equipped with a simple browser-based database administration interface, the **DBA**. It allows you to create and administrate databases, evaluate queries in realtime, view log files and manage users. The server-side code is completely written in XQuery and RESTXQ.

These were our design goals:

- The code base is supposed to help and motivate you developing your own RESTXQ web applications.
- The whole DBA code consumes only 100 KB. It uses very simple Javascript code that should run with nearly every browser. The interface is functional, but limited in terms of flashiness and interactivity.
- We tried to make the DBA features as self-explanatory as possible. All functionalities are also available via Commands, XQuery Modules or the Java GUI.
- The `dba` sub-directory can simply be copied and moved to any other place. All URL paths point to the same directory; it should be straightforward to adjust the RESTXQ path.

Please be aware that it is an obvious **security risk** if you simply put DBA online along with your web page. At the very least, you should change the DBA path and the password of the `admin` user!

The DBA has just been released, and it is in beta stage. We are looking forward to your reports and feature requests! Patches and extensions are even more welcome.

Startup

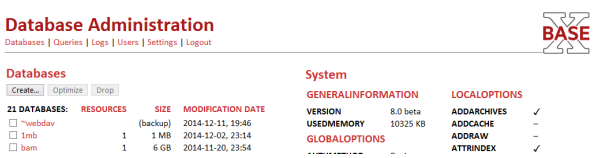
- Download the **ZIP Archive** or the **Windows Installer** from the download page
- Start the BaseX HTTP Server
- Open a browser and visit the URL `http://localhost:8984/dba`

First Steps

On the welcome page, you will need to authenticate yourself by entering a username and password. The default user is `admin`/`admin`. By specifying a host and port, you can communicate with a remote BaseX server instance. If you keep the field empty, you will be connected to the local BaseX instance.

The main page of the DBA interface contains a list of all databases on the left. On the right, the global and local options are listed, along with some system information.

With the "Create…" button, a new database can be created. Existing database can be viewed, optimized, and dropped.



DBA Main Page

Changelog

Introduced with Version 8.0.

Part III. General Info

Chapter 12. Databases

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section.

In BaseX, a *database* is a pretty light-weight concept and can be compared to 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.

Note: A main-memory only database can be created using the the `SET MAINMEM true` command before calling `CREATE DB` (see below for more).

Access Resources

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

XML Documents

Various XQuery functions exist to access XML documents in databases:

Function	Example	Description
`db:open`	`db:open("db", "path/to/docs")`	Returns all documents that are found in the database `db` at the (optional) path `path/to/docs`.
`fn:collection`	`collection("db/path/to/docs")`	Returns all documents at the location `path/to/docs` in the database `db`.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 database will be returned that has been opened in the global context.
`fn:doc`	`doc("db/path/to/doc.xml")`	Returns the document at the location `path/to/docs` in the database `db`.An error is raised if the specified yields zero or more than one document.

You can access multiple databases in a single query:

```
for $i in 1 to 100
return db:open('books' || $i)//book/title
```

If the DEFAULTDB option is turned on, the path argument of the `fn:doc` or `fn:collection` function will first be resolved against the globally opened database.

Two more functions are available for retrieving information on database nodes:

Function	Example	Description
`db:name`	`db:name($node)`	Returns the name of the database in which the specified `$node` is stored.
`db:path`	`db:path($node)`	Returns the path of the database document in which the specified `$node` is stored.

The `fn:document-uri` and `fn:base-uri` functions return URIs that can also be reused as arguments for the `fn:doc` and `fn:collection` functions. As a result, the following example query always returns `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.
- `doc("myfile.xml")` : retrieves the given file from the file system and returns it as a main-memory document node. Note that updates to main-memory nodes are not automatically written back to disk unless the `WRITEBACK` option is set.
- `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`.
- Existing resources can be replaced 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 ADDCACHE option will first cache the input before adding it to the database. This is helpful when the input documents to be added are expected to eat up too much main memory.

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
SET ADDCACHE true
ADD /path/to/xml/documents
STORE TO images/ 123.jpg
FLUSH
DELETE /
```

You may as well use the BaseX-specific XQuery Database Functions to create, 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

In Memory Database

- In the standalone context, a main-memory database can be created (using `CREATE DB`), which can then be accessed by subsequent commands.
- If a BaseX server instance is started, and if a database is created in its context (using `CREATE DB`), other BaseX client instances can access (and update) this database (using OPEN, db:open, etc.) as long as no other database is opened/created by the server.

Note: main-memory database instances are also created by the invocation of `doc(...)` or `collection(...)`, if the argument is not a database (no matter which value is set for MAINMEM). In other words: the same internal representation is used for main-memory databases and documents/collections generated via XQuery.

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.

Chapter 13. Binary Data

Read this entry online in the BaseX Wiki.

This page is linked from the Database page.

The BaseX store also provides support for *raw files* (binary data). A database may contain both XML documents and raw files. 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 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.

Chapter 14. Parsers

Read this entry online in the BaseX Wiki.

This article is part of the Getting Started Section. It presents the available parsers that can be used to import various data sources in BaseX databases. Please visit the Serialization article if you want to know how to export data.

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 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 and db:replace functions can also be used to add new XML documents to the database. The following example query uses the internal XML 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, 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

Downloads

TagSoup is already included in the full BaseX distributions (`BaseX.zip`, `BaseX.exe`, etc.). It can also be manually downloaded and embedded on the appropriate platforms.

Maven

An easy way to add TagSoup to your own project is to follow this steps:

1. visit MVN TagSoup Repository

2. click on the version you want

3. you can see on the first tab called Maven a XML like this :

```
<dependency>
  <groupId>org.ccil.cowan.tagsoup</groupId>
  <artifactId>tagsoup</artifactId>
  <version>1.2.1</version>
</dependency>
```

4. copy that in your own maven project's pom.xml under the <dependencies> tag.

5. don't forget to run `mvn jetty:run` again

Debian

With Debian, TagSoup will be automatically detected and included after it has been installed via:

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

TagSoup Options

TagSoup offers a variety of options to customize the HTML conversion. For the complete list please visit the TagSoup 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 HTMLPARSER
method=xml,nons=true,ncdata=true,nodefaults=true,nobogons=true,nocolons=true,ignorable=true
SET CREATEFILTER *.html
```

XQuery

The HTML Module provides a function for converting HTML to XML documents.

Documents can also be converted by specifying the parser and additional options in the query prolog:

```
declare option db:parser "html";
declare option db:htmlparser "html=false,nodefaults=true";
doc("index.html")
```

JSON Parser

BaseX can also import JSON documents. The resulting format is described in the documentation for the XQuery JSON Module:

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

XQuery

The JSON Module provides functions for converting JSON objects to XML documents.

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` or `space` or an arbitrary character.
- **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. Unicode code points can be specified as separators; `32` is the code point for spaces:

```
SET PARSER csv
SET CSVPARSER encoding=utf-8, lines=true, header=false, separator=space
SET CREATEFILTER *.csv
```

XQuery

The CSV Module provides a function for converting CSV to XML documents.

Documents can also be converted by specifying the parser in the query prolog. The following example query adds all CSV files that are located in the directory `2Bimported` to the database `DB` and interprets the first lines as column headers:

```
declare option db:parser "csv";
declare option db:csvparser "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 TEXTPARSER lines=yes
SET CREATEFILTER *
```

XQuery

Similar to the other formats the text parser can be specified in the prolog of an XQuery expression:

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

Changelog

Version 7.8

- Updated: parser options

Version 7.7.2

- Removed: CSV option "format"

Version 7.3

- Updated: the CSV `SEPARATOR` option may now be assigned arbitrary single characters

Version 7.2

- Updated: Enhanced support for TagSoup options

Chapter 15. Commands

Read this entry online in the BaseX Wiki.

This article is part of the Getting Started Section. It lists all database commands supported by BaseX. Commands can e.g. be executed from the Command Line, Scripts, the Clients, REST, the input field in the GUI and 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.

Basics

Command Scripts

Database commands in both the string and XML syntax can be placed in a text file and stored on disk. The default extension for BaseX command scripts is `.bxs`. If the path to a command script is passed on to BaseX, it will automatically be recognized and evaluated as such.

String Syntax

Multiple commands can be written in a single line and separated by semicolons, or stored as command script. Lines starting with # are interpreted as comments and are skipped. The following script creates a database, adds two documents to it and performs a query:

```
CREATE DB test
ADD input.xml
ADD TO embedded.xml <root>embedded</root>
# run query
XQUERY count(//text())
```

XML Syntax

The string syntax is limited when XML snippets need to be embedded in a command, or when complex queries are to be specified.

This is why database commands can also be specified in XML. Multiple commands can be enclosed by a `<commands/>` root element:

```
<commands>
  <create-db name='test'/>
  <add>input.xml</add>
  <add path='embedded.xml'><root>embedded</root></add>
  <xquery>count(//text())</xquery>
</commands>
```

Glob Syntax

Some commands support the glob syntax 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

Database and user names follow the same naming constraints: Names are restricted to ASCII characters. They must at least have one character, and they may contain letters, numbers and any of the special characters `!#$%&'()+-=@[]^_`{}~`. The following characters are reserved for other features:

- `,?*` : glob syntax
- `;` : Separator for multiple database commands on the command line
- `\/` : Directory path separators
- `.` : hidden folders (e.g. the .logs directory)
- `:*?\"<>|}` : invalid filename characters on Windows

Aliases

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

Database Operations

CREATE DB

Syntax	`CREATE DB [name] ([input])`
XML Syntax	`<create-db name='...'>([input])</create-db>`
Permission	*CREATE*
Summary	Creates a new database with the specified `name` and, optionally, an initial `input`, and opens it. An existing database will be overwritten.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 • several additional Create Options can be set to influence the creation process.
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 DB input` creates an empty database `input`. • `CREATE DB xmark http://files.basex.org/xml/xmark.xml` 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 DB input input.xml` creates a database `input` with `input.xml` as initial document, which will be parsed with Java's default XML parser. • `<create-db name='simple'><hello>Universe</hello></create-db>` creates a database named `simple` with an initial document `<hello>Universe</hello>`.

OPEN

Syntax	`OPEN [name] ([path])`
XML Syntax	`<open name='...' (path='...')/>`
Permission	*READ*
Summary	Opens the database specified by `name`. The documents to be opened can be specified by the [path] argument.

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

CHECK

Syntax	`CHECK [input]`
XML Syntax	`<check input='...'/>`
Permission	*READ/CREATE*
Summary	This convenience command combines OPEN and CREATE DB: if a database with the name `input` exists, it is opened. Otherwise, a new database is created; if the specified input points to an existing resource, it is stored as initial content.
Errors	The command fails if the addressed database could neither be opened nor created.

CLOSE

Syntax	`CLOSE`
XML Syntax	`<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

Syntax	`EXPORT [path]`
XML Syntax	`<export path='...'/>`
Permission	*CREATE*
Summary	Exports all documents in the database to the specified file `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

Syntax	`CREATE INDEX [TEXT|ATTRIBUTE|FULLTEXT]`
XML Syntax	`<create-index type='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

Syntax	`DROP INDEX [TEXT|ATTRIBUTE|FULLTEXT]`
XML Syntax	`<drop-index type='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 DB

Syntax	`ALTER DB [name] [newname]`
XML Syntax	`<alter-db 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 DB db tempdb` renames the database `db` into `tempdb`.

DROP DB

Syntax	`DROP DB [name]`
XML Syntax	`<drop-db 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

Syntax	`CREATE BACKUP [name]`
XML Syntax	`<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-2014-04-01-12-27-28.zip`) in the database directory.

RESTORE

Syntax	`RESTORE [name]`
XML Syntax	`<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.

INSPECT

Syntax	`INSPECT`
XML Syntax	`<inspect/>`

Permission	*READ*
Summary	Performs some integrity checks on the opened database and returns a brief summary.

DROP BACKUP

Syntax	`DROP BACKUP [name]`
XML Syntax	`<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

Syntax	`SHOW BACKUPS`
XML Syntax	`<show-backups/>`
Permission	*CREATE*
Summary	Shows all database backups.

COPY

Syntax	`COPY [name] [newname]`
XML Syntax	`<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 DB

Syntax	`INFO DB`
XML Syntax	`<info-db/>`
Permission	*READ*
Summary	Shows information on the currently opened database.
Errors	The command fails if no database is opened.

INFO INDEX

Syntax	`INFO INDEX ([TAG|ATTNAME|PATH|TEXT|ATTRIBUTE|FULLTEXT])`
XML Syntax	`<info-index type='tag|attname|path|text|attribute|fulltext'/>`
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

Syntax	`INFO STORAGE [start end] | [query]`
XML Syntax	`<info-storage>([query])</info-storage>`

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

Syntax	`LIST ([name] ([path]))`
XML Syntax	`<list (name='...' (path='...'))/>`
Permission	*NONE*
Summary	Lists all available databases. If `name` is specified, the resources of a database are listed. The output can be further restricted to the resources matching the specified `path`.
Errors	The command fails if the optional database cannot be opened, or if the existing databases cannot be listed for some other reason.

XQUERY

Syntax	`XQUERY [query]`
XML Syntax	`<xquery>[query]</xquery>`
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

Syntax	`RETRIEVE [path]`
XML Syntax	`<retrieve path='...'/>`
Permission	*READ*
Summary	Retrieves a raw file from the opened database at the specified `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.

FIND

Syntax	`FIND [query]`
XML Syntax	`<find>[query]</find>`
Permission	*READ*
Summary	Builds and runs a query for the specified `query` terms. 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.

TEST

Syntax	`TEST [path]`
XML Syntax	`<test path='...'/>`
Permission	*ADMIN*
Summary	Runs all XQUnit tests in the specified `path`. The path can point to a single file or a directory.Unit testing can also be triggered via `-t` on command line.
Errors	The command fails if at least one test fails.
Examples	• `TEST project/tests` runs all tests in the directory `project/tests`.

REPO INSTALL

Syntax	`REPO INSTALL [path]`
XML Syntax	`<repo-install path='...'/>`
Permission	*CREATE*
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

Syntax	`REPO LIST`
XML Syntax	`<repo-list/>`
Permission	*READ*
Summary	Lists all installed packages.

REPO DELETE

Syntax	`REPO DELETE [name]`
XML Syntax	`<repo-delete name='...'/>`
Permission	*CREATE*
Summary	Deletes the package with name `name`, optionally followed by a version.
Errors	The command fails if the package to be deleted participates in a dependency.

Updates

ADD

Syntax	`ADD (TO [path]) [input]`

XML Syntax	`<add (path='...')>[input]</add>`
Permission	*WRITE*
Summary	Adds a file, 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. • A document with the same path may occur than once in a database. If this is unwanted, REPLACE can be used. • If a file is too large to be added in one go, its data structures will be cached to disk first. Caching can be enforced by turning the ADDCACHE option on. If files are to be added to an empty database, it is usually faster to use the CREATE DB command and specify the initial input as argument.
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

Syntax	`DELETE [path]`
XML Syntax	`<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

Syntax	`RENAME [path] [newpath]`
XML Syntax	`<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

Syntax	`REPLACE [path] [input]`
XML Syntax	`<replace path='...'>[input]</replace>`
Permission	*WRITE*
Summary	Replaces a document in the currently opened database, addressed by `path`, with the file or XML string specified by `input`, or adds a new document if the resource does not exist yet.

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

Syntax	`STORE (TO [path]) [input]`
XML Syntax	`<store (path='...')>[input]</store>`
Permission	*WRITE*
Summary	Stores a raw file in the opened database 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

Syntax	`OPTIMIZE (ALL)`
XML Syntax	`<optimize/>` `<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

Syntax	`FLUSH`
XML Syntax	`<flush/>`
Permission	*WRITE*
Summary	Explicitly flushes the buffers of the currently opened database to disk. This command is applied if AUTOFLUSH has been set to `false`.
Errors	The command fails if no database is opened.

Server Administration

SHOW SESSIONS

Syntax	`SHOW SESSIONS`
XML Syntax	`<show-sessions/>`
Permission	*ADMIN*
Summary	Shows all sessions that are connected to the current server instance.

SHOW USERS

Syntax	`SHOW USERS (ON [database])`

XML Syntax	`<show-users (database='...')/>`
Permission	*ADMIN*
Summary	Shows all users that are registered in the database. If a `database` is specified, all user will be shown for which a pattern was specified that matches the database name.
Errors	The command fails if the optional database could not be opened.

KILL

Syntax	`KILL [target]`
XML Syntax	`<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.

User Management

CREATE USER

Syntax	`CREATE USER [name] ([password])`
XML Syntax	`<create-user name='...'>([password])</create-user>`
Permission	*ADMIN*
Summary	Creates a user with the specified `name` and `password`. If no password is specified, it is requested via the chosen frontend (GUI or bash).
Errors	The command fails if the specified user already exists.

ALTER USER

Syntax	`ALTER USER [name] ([newname])`
XML Syntax	`<alter-user name='...' newname='...'/>`
Permission	*ADMIN*
Summary	Renames the user with the specified `name` to `newname`.
Errors	The command fails if the specified user does not exist, or if the new user already exists.

ALTER PASSWORD

Syntax	`ALTER PASSWORD [name] ([password])`
XML Syntax	`<alter-password name='...'>([password])</alter-password>`
Permission	*ADMIN*
Summary	Alters the `password` of the user with the specified `name`. If no password is specified, it is requested via the chosen frontend (GUI or bash).
Errors	The command fails if the specified user does not exist.

DROP USER

Syntax	`DROP USER [name] (ON [pattern])`:
XML Syntax	`<drop-user name='...' (pattern='...')/>`

Permission	*ADMIN*
Summary	Drops the user with the specified `name`. The Glob Syntax can be used to address more than one database or user. If a glob `pattern` is specified, only the pattern will be removed.
Errors	The command fails if `admin` is specified as user name, or if the specified user does not exist or is currently logged in.

GRANT

Syntax	`GRANT [NONE|READ|WRITE|CREATE|ADMIN] (ON [pattern]) TO [user]`
XML Syntax	`<grant name='...' permission='none|read|write|create|admin' (pattern='...')/>`
Permission	*ADMIN*
Summary	Grants the specified permission to the specified `user`. The Glob Syntax can be used to address more than one user. If a glob `pattern` is specified, the permission will be applied to all databases that match this pattern.
Errors	The command fails if `admin` is specified as user name or if the specified user does not exist.
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

Syntax	`PASSWORD ([password])`
XML Syntax	`<password>([password])</password>`
Permission	*NONE*
Summary	Changes the `password` of the current user. If no password is specified, it is requested via the chosen frontend (GUI or bash).

General Commands

RUN

Syntax	`RUN [file]`
XML Syntax	`<run file='...'/>`
Permission	*depends on input*
Summary	Evaluates the contents of `file` as XQuery expression. If the file ends with the suffix `.bxs`, the file content will be evaluated as command script. This command can be used to run several commands in a single transaction.
Errors	The command fails if the specified file does not exist, or if the retrieved input is invalid. It will be canceled as soon as one of the executed commands fails.
Examples	• `RUN query.xq` will evaluated the specified file as XQuery expression • `RUN commands.bxs` will evaluated the specified file as command script

EXECUTE

Syntax	`EXECUTE [input]`
XML Syntax	`<execute>[input]</execute>`
Permission	*depends on input*

Summary	Evaluates the specified `input` as command script. This command can be used to run several commands in a single transaction.
Errors	The command fails if the syntax of the specified input is invalid. It will be canceled as soon as one of the executed commands fails.
Examples	• `EXECUTE "create db db1; create db db2"` • `EXECUTE "<commands><create-db name='db1'/><create-db name='db2'/><</commands>"` both commands will create two databases `db1` and `db2` in a single transaction.

GET

Syntax	`GET [option]`
XML Syntax	`<get option='...'/>`
Permission	*NONE*
Summary	Returns the current value of the Option specified via `option`. Global options can only be requested by users with ADMIN permissions.
Errors	The command fails if the specified option is unknown.

SET

Syntax	`SET [option] ([value])`
XML Syntax	`<set option='...'>([value])</set>`
Permission	*NONE*
Summary	Sets the Option specified by `option` to a new `value`. Only local options can be modified. 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

Syntax	`INFO`
XML Syntax	`<info/>`
Permission	*READ*
Summary	Shows global information.

HELP

Syntax	`HELP ([command])`
XML Syntax	`<help>([command])</help>`
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

Syntax	`EXIT`
XML Syntax	`<exit/>`
Permission	*NONE*

Summary	Exits the console mode.

QUIT

Syntax	`QUIT`
XML Syntax	`<quit/>`
Permission	*NONE*
Summary	Exits the console mode (alias of EXIT).

Changelog

Version 8.2

- Removed: `CREATE EVENT`, `DROP EVENT` and `SHOW EVENTS` command

Version 8.0

- Updated: commands for User Management
- Updated: `OPEN`: path argument added
- Removed: `CS` command
- Added: `QUIT`

Version 7.9

- Added: `TEST` runs XQUnit tests.

Version 7.7

- Updated: syntax of valid names.

Version 7.5

- Added: `EXECUTE` executes a command script.
- Added: `INSPECT` performs integrity checks.
- Added: automatic detection of Command Scripts.
- Removed: `SHOW DATABASES`; information is also returned by `SHOW SESSIONS`.
- Removed: `OPEN`: path argument.

Version 7.3

- Added: XML Syntax added.
- Updated: `CHECK` can now be used to create empty databases.
- Updated: Names and paths in `OPEN` and `LIST` are now specified as separate arguments.

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.

Chapter 16. Options

Read this entry online in the BaseX Wiki.

This page is linked from the Getting Started Section.

The options listed on this page influence the way how database commands are executed and XQuery expressions are evaluated. Options are divided into **global options** , which are valid for all BaseX instances, and **local options**, which are specific to a client or session. Values of options are either *strings*, *numbers* or *booleans*.

The `.basex` configuration file is parsed by every new local BaseX instance. It contains all global options and, optionally, local options at the end of the file.

Various ways exist to access and change options:

- The current value of an option can be requested with the GET command. Local options can be changed via SET. All values are *static*: They stay valid until they are changed once again by another operation. If an option is of type *boolean*, and if no value is specified, its current value will be inverted.
- Initial values for global options can also be specified via system properties, which can e.g. be passed on with the -D flag on command line, or using System.setProperty() before creating a BaseX instance. The specified keys need to be prefixed with `org.basex.`. An example:

```
java -Dorg.basex.CHOP=false -cp basex.jar org.basex.BaseX -c"get chop"
CHOP: false
```

- Local options can also be set in the prolog of an XQuery expression. In the option declaration, options need to be bound to the Database Module namespace. All values will be reset after the evaluation of a query:

```
declare option db:chop 'false';
...
```

- Beside that, local options can be applied locally by using pragmas:

```
(# db:chop false #) { parse-xml('<xml> hi </xml>') }
```

If options are implicitly changed by operations in the GUI, the underlying commands will be listed in the Info View.

Global Options

Global options are constants. They can only be set in the configuration file or via system properties (see above). One exception is the DEBUG option, which can also be changed at runtime by users with admin permissions.

General

DEBUG

Signature	`DEBUG [boolean]`
Default	`false`
Summary	Sends internal debug info to STDERR. This option can be turned on to get additional information for development and debugging purposes. It can also be triggered on command line via `-d`.

DBPATH

Signature	`DBPATH [path]`

Default	`{home}/BaseXData` or `{home}/data`
Summary	Points to the directory in which all databases are located.

REPOPATH

Signature	`REPOPATH [path]`
Default	`{home}/BaseXRepo`
Summary	Points to the Repository, in which all XQuery modules are located.

LANG

Signature	`LANG [language]`
Default	`English`
Summary	Specifies the interface language. Currently, seven languages are available: 'English', 'German', 'French', 'Dutch', 'Italian', 'Japanese', and 'Vietnamese'.

LANGKEY

Signature	`LANGKEY [boolean]`
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.

GLOBALLOCK

Signature	`GLOBALLOCK [boolean]`
Default	`false`
Summary	Controls if local (database) or global (process) locking will be used for managing read and write operations. The article on Transaction Management provides more details on concurrency control.

Client/Server Architecture

HOST

Signature	`HOST [host]`
Default	`localhost`
Summary	This host name is used by the client when connecting to a server. This option can also be changed when running the client on command line via `-n`.

PORT

Signature	`PORT [port]`
Default	`1984`
Summary	This port is used by the client when connecting to a server. This option can also be changed when running the client on command line via `-p`.

SERVERPORT

Signature	`SERVERPORT [port]`
Default	`1984`

Summary	This is the port the database server will be listening to. This option can also be changed when running the server on command line via `-p`.

USER

Signature	`USER [name]`
Default	*empty*
Summary	Represents a user name, which is used for accessing the server or an HTTP service: • The default value will be overwritten if a client specifies its own credentials. • If the default value is empty, login will only be possible if the client specifies credentials. • The option can also be changed on command line via `-U`.

PASSWORD

Signature	`PASSWORD [password]`
Default	*empty*
Summary	Represents a password, which is used for accessing the server or an HTTP service: • The default value will be overwritten if a client specifies its own credentials. • If the default value is empty, login will only be possible if the client specifies credentials. • The option can also be changed on command line via `-P`. • Please note that it is a security risk to specify your password in plain text.

AUTHMETHOD

Signature	`AUTHMETHOD [method]`
Default	*Basic*
Summary	Specifies the HTTP Authentication, which will be proposed by the HTTP server if a client sends an unauthorized request. Allowed values are `Basic` and `Digest`.

SERVERHOST

Signature	`SERVERHOST [host|ip]`
Default	*empty*
Summary	This is the host name or ip address the server is bound to. If the option is set to an empty string (which is the default), the server will be open to all clients.

PROXYHOST

Signature	`PROXYHOST [host]`
Default	*empty*
Summary	This is the host name of a proxy server. If the value is an empty string, it will be ignored.

PROXYPORT

Signature	`PROXYPORT [port]`
Default	`0`
Summary	This is the port number of a proxy server. If the value is set to `0`, it will be ignored.

NONPROXYHOSTS

Signature	`NONPROXYHOSTS [hosts]`
Default	*empty*
Summary	This is a list of hosts that should be directly accessed. If the value is an empty string, it will be ignored.

IGNORECERT

Signature	`IGNORECERT [boolean]`
Default	`false`
Summary	This option can be turned on to ignore untrusted certificates when connecting to servers. Please use this option carefully.

TIMEOUT

Signature	`TIMEOUT [seconds]`
Default	`30`
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`. It is ignored for `ADMIN` operations.

KEEPALIVE

Signature	`KEEPALIVE [seconds]`
Default	`600`
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`.

PARALLEL

Signature	`PARALLEL [number]`
Default	`8`
Summary	Denotes the maximum allowed `number` of parallel transactions.Note that a higher number of parallel operations may increase disk activity and thus slow down queries. In some cases, a single transaction may even give you better results than any parallel activity.

LOG

Signature	`LOG [boolean]`
Default	`true`
Summary	Turns Logging of server operations and HTTP requests on/off. This option can also be changed when running the server on command line via `-z`.

LOGMSGMAXLEN

Signature	`LOGMSGMAXLEN [length]`
Default	`1000`
Summary	Specifies the maximum length of a single log message.

HTTP Options

If BaseX is run as web servlet, the HTTP options must be specified in the `webapp/WEB-INF` directory and the `jetty.xml` and `web.xml` configuration files.

WEBPATH

Signature	`WEBPATH [path]`
Default	`{home}/BaseXWeb` or `{home}/webapp`
Summary	Points to the directory in which all the Web Application contents are stored, including XQuery, Script, RESTXQ and configuration files. This option is ignored if BaseX is deployed as web servlet.

RESTXQPATH

Signature	`RESTXQPATH [path]`
Default	*empty*
Summary	Points to the directory which contains the RESTXQ modules of a web application. Relative paths will be resolved against the WEBPATH directory.

RESTPATH

Signature	`RESTPATH [path]`
Default	*empty*
Summary	Points to the directory which contains XQuery files and command scripts, which can be evaluated via the REST run operation. Relative paths will be resolved against the WEBPATH directory.

HTTPLOCAL

Signature	`HTTPLOCAL [boolean]`
Default	`false`
Summary	By default, if BaseX is run as Web Application, a database server instance will be started as soon as the first HTTP service is called. The server can then be addressed by other BaseX clients in parallel to the HTTP services.If the option is set to `false`, the database server will be disabled.

STOPPORT

Signature	`STOPPORT [port]`
Default	`8985`
Summary	This is the port on which the HTTP Server can be locally closed: • The listener for stopping the web server will only be started if the specified value is greater than `0`. • The option is ignored if BaseX is used as a Web Application or started via Maven. • This option can also be changed when running the HTTP server on command line via `-s`.

Create Options

General

MAINMEM

Signature	`MAINMEM [boolean]`

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.

ADDCACHE

Signature	`ADDCACHE [boolean]`
Default	`false`
Summary	If this option is activated, data structures of documents will first be cached to disk before being added to the final database. This option is helpful when larger documents need to be added, and if the existing heuristics cannot estimate the input size (e.g. when adding directories or sending input streams).

Parsing

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

SKIPCORRUPT

Signature	`SKIPCORRUPT [boolean]`
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 [boolean]`
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 is not found in the classpath.

CSVPARSER

Signature	`CSVPARSER [options]`
Default	*empty*
Summary	Specifies the way how CSV data is to be parsed. The available options are listed in the CSV Module.

JSONPARSER

Signature	`JSONPARSER [options]`
Default	*empty*
Summary	Specifies the way how JSON data is to be parsed. The available options are listed in the JSON Module.

TEXTPARSER

Signature	`TEXTPARSER [options]`
Default	*empty*
Summary	Specifies the way how TEXT data is to be parsed. Available options are listed in the article on Parsers.

XML Parsing

CHOP

Signature	`CHOP [boolean]`
Default	`true`

Summary	Many XML documents include whitespaces that have been added to improve readability. The `CHOP` option controls the white-space processing mode of the XML parser:
	• By default, this option is set to `true`. This way, leading and trailing whitespaces from text nodes will be chopped and all empty text nodes will be discarded.
	• The flag should be turned off if a document contains mixed content.
	• The flag can also be turned off on command line via `-w`.
	• If the `xml:space="preserve"` attribute is attached to an element, chopping will be turned off for all descendant text nodes. In the following example document, the whitespaces in the text nodes of the `text` element will not be chopped:
	`<xml>`
	`  <title>`
	`  Demonstrating the CHOP flag`
	`  </title>`
	`  <text xml:space="preserve">To <b>be</b>, or not to <b>be</b>, that is the question.</text>`
	`</xml>`

STRIPNS

Signature	`STRIPNS [boolean]`
Default	`false`
Summary	Strips all namespaces from an XML document and all elements while parsing.

INTPARSE

Signature	`INTPARSE [boolean]`
Default	`false`
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 [boolean]`
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.

XINCLUDE

Signature	`XINCLUDE [boolean]`
Default	`true`
Summary	Resolves XInclude inclusion tags and merges referenced XML documents. By default, this option is switched to `true`. This option is only available if the standard Java XML Parser is used (see INTPARSE).

CATFILE

Signature	`CATFILE [path]`
Default	*empty*
Summary	Specifies a catalog file to locally resolve DTDs; see the entry on Catalog Resolvers for more details.

Indexing

The current index and full-text index options will be stored in a new database, and take effect if indexes are rebuilt via the OPTIMIZE.

TEXTINDEX

Signature	`TEXTINDEX [boolean]`
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 [boolean]`
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 [boolean]`

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 [int]`
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 [int]`
Default	`100`
Summary	Specifies the maximum number of distinct values (categories) that will be stored together with the element/attribute names or unique paths in the Name Index or Path Index. The value of this option will be assigned once to a new database, and cannot be changed after that.

UPDINDEX

Signature	`UPDINDEX [boolean]`
Default	`false`
Summary	If turned on, incremental indexing will be applied to new databases: • With each update, the text and attributes indexes will be refreshed as well. • The advantage is that the value index structures will always be up-to-date. • However, updates will usually take longer (the article on Index Structures provides more details). • The value of this option will be assigned once to a new database. It can be reassigned by running OPTIMIZE ALL or db:optimize($db, true()).

AUTOOPTIMIZE

Signature	`AUTOOPTIMIZE [boolean]`
Default	`false`
Summary	If turned on, auto optimization will be applied to new databases: • With each update, outdated indexes and database statistics will be recreated. • As a result, the index structures will always be up-to-date. • However, updates can take much longer, so this option should only be activated for medium-sized databases. • The value of this option will be assigned once to a new database. It can be reassigned by running OPTIMIZE or db:optimize.

INDEXSPLITSIZE

Signature	`INDEXSPLITSIZE [num]`
Default	`0`

Summary	This option affects the construction of new text and attribute indexes. It specifies the number of index build operations that are performed before writing partial index data to disk. By default, if the value is set to 0, some dynamic split heuristics are applied. By setting the value to its maximum (2147483647), the index will never be split.

FTINDEXSPLITSIZE

Signature	`FTINDEXSPLITSIZE [num]`
Default	`0`
Summary	This option affects the construction of new full-text indexes. It specifies the number of index build operations that are performed before writing partial index data to disk. By default, if the value is set to 0, some dynamic split heuristics are applied. By setting the value to its maximum (2147483647), the index will never be split.

Full-Text

STEMMING

Signature	`STEMMING [boolean]`
Default	`false`
Summary	If `true`, all tokens will be stemmed during full-text indexing, using a language-specific stemmer implementation. By default, token will not be stemmed.

CASESENS

Signature	`CASESENS [boolean]`
Default	`false`
Summary	If `true`, the case of tokens will be preserved during full-text indexing. By default, case will be ignored (all tokens will be indexed in lower case).

DIACRITICS

Signature	`DIACRITICS [boolean]`
Default	`false`
Summary	If set to `true`, diacritics will be preserved during full-text indexing. By default, diacritics will be removed.

LANGUAGE

Signature	`LANGUAGE [lang]`
Default	`en`
Summary	The specified language will influence the way how an input text will be tokenized. This option is mainly important if tokens are to be stemmed, or if the tokenization of a language differs from Western languages.

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 or available online at http://files.basex.org/etc/stopwords.txt.

Query Options

QUERYINFO

Signature	`QUERYINFO [boolean]`
Default	`false`
Summary	Prints more information on internal query rewritings, optimizations, and performance. By default, this info is shown in the Info View in the GUI. It can also be activated on command line via `-V`.

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.

MIXUPDATES

Signature	`MIXUPDATES`
Default	`false`
Summary	Allows queries to both contain updating and non-updating expressions. All updating constraints will be turned off, and nodes to be returned will be copied before they are modified by an updating expression. – By default, this option is set to `false`, because the XQuery Update Facility does not allow to return results.

BINDINGS

Signature	`BINDINGS [vars]`
Default	*empty*
Summary	Contains external variables to be bound to a query. The string must comply with the following rules: • Variable names and values must be separated by equality signs. • Multiple variables must be delimited by commas. • Commas in values must be duplicated. • Variables may optionally be introduced with a leading dollar sign. • If a variable uses a namespace different to the default namespace, it can be specified with the Clark Notation or Expanded QName Notation. This option can also be used on command line with the flag `-b`.

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` binds the value `x` to the variable $a with the namespace `URI`.
	• In the following Command Script, the value `hello world!` is bound to the variable $GREETING:
	`SET BINDINGS GREETING="hello world!"`
	`XQUERY declare variable $GREETING external; $GREETING`

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.

INLINELIMIT

Signature	`INLINELIMIT`
Default	`100`
Summary	The XQuery compiler inlines functions to speed up query evaluation. Inlining will only take place if a function body is not too large (i.e., if it does not contain too many expressions). With this option, this maximum number of expressions can be specified.Function inlining can be turned off by setting the value to `0`. The limit can be locally overridden via the `%basex:inline` annotation.

TAILCALLS

Signature	`TAILCALLS`
Default	`256`
Summary	Specifies how many stack frames of tail-calls are allowed on the stack at any time. When this limit is reached, tail-call optimization takes place and some call frames are eliminated. The feature can be turned off by setting the value to `-1`.

DEFAULTDB

Signature	`DEFAULTDB`
Default	`false`
Summary	If this option is turned on, paths specified in the `fn:doc` and `fn:collections` functions will first be resolved against a database that has been opened in the global context outside the query (e.g. by the OPEN command). If the path does not match any existing resources, it will be resolved as described in the article on accessing database resources.

CACHEQUERY

Signature	`CACHEQUERY [boolean]`
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 [boolean]`
Default	`false`
Summary	By activating this option, the XQuery `doc()` and `collection()` functions will create database instances for the addressed input files.

CHECKSTRINGS

Signature	`CHECKSTRINGS [boolean]`
Default	`true`

Summary	If this option is turned off, strings from external sources will be adopted as is, i. e., without being checked for valid XML characters: • This option affects Java Bindings and the string conversion and input functions archive:create, archive:extract-text, archive:update, convert:binary-to-string, fetch:text, file:read-text, and zip:text-entry. • Please be aware that an inconsiderate use of this option may cause unexpected behavior when storing or outputting strings.

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.

RUNQUERY

Signature	`RUNQUERY [boolean]`
Default	`true`
Summary	Specifies if a query will be executed or parsed only. This option can also be changed on command line via `-R`.

RUNS

Signature	`RUNS [num]`
Default	`1`
Summary	Specifies how often a query will be evaluated. The result is serialized only once, and the measured times are averages of all runs. This option can also be changed on command line via `-r`.

Serialization Options

SERIALIZE

Signature	`SERIALIZE [boolean]`
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`. It can also be turned off on command line via `-z`.

SERIALIZER

Signature	`SERIALIZER [params]`
Default	*empty*
Summary	Parameters for serializing query results. The string must comply with the following rules: • Variable names and values must be separated by equality signs. • Multiple variables must be delimited by commas. • Commas in values must be duplicated. The option can also be used on command line with the flag `-s`.

Examples	• `encoding=US-ASCII,omit-xml-declaration=no` : sets the encoding to `US-ASCII` and prints the XML declaration. • `item-separator=,,` : separates serialized items by a single comma.

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.

XMLPLAN

Signature	`XMLPLAN [boolean]`
Default	`false`
Summary	Prints the execution plan of an XQuery expression in its XML representation. This option can also be activated on command line via `-x`.

COMPPLAN

Signature	`COMPPLAN [boolean]`
Default	`true`
Summary	Generates the query plan, which can be activated via [[#XMLPLAN|XMLPLAN], before or after query compilation. This option can also be activated on command line via `-X`.

DOTPLAN

Signature	`DOTPLAN [boolean]`
Default	`false`
Summary	Visualizes the execution plan of an XQuery expression with dotty and saves its dot file in the query directory.

DOTCOMPACT

Signature	`DOTCOMPACT [boolean]`
Default	`false`
Summary	Chooses a compact dot representation.

Other Options

AUTOFLUSH

Signature	`AUTOFLUSH [boolean]`
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.

WRITEBACK

Signature	`WRITEBACK [boolean]`

Default	`false`
Summary	Propagates updates on main-memory instances of files that have been retrieved via `fn:doc` or `fn:collection` back to disk. This option can also be activated on command line via `-u`. Please note that, when turning this option on, your original files will not be backed up.

MAXSTAT

Signature	`MAXSTAT [num]`
Default	`30`
Summary	Specifies the maximum number of index occurrences printed by the `INFO INDEX` command.

Changelog

Version 8.2

- Removed: `EVENTPORT`

Version 8.1

- Added: `IGNORECERT`, `RESTPATH`

Version 8.0

- Added: `MIXUPDATES`, `AUTOOPTIMIZE`, `AUTHMETHOD`, `XINCLUDE`
- Updated: `PROXYPORT`: default set to 0; will be ignored. `PROXYHOST`, `NONPROXYHOSTS`: empty strings will be ignored.

Version 7.8.1

- Updated: `ADDARCHIVES`: parsing of TAR and TGZ files.

Version 7.8

- Added: `CSVPARSER`, `JSONPARSER`, `TEXTPARSER`, `HTMLPARSER`, `INLINELIMIT`, `TAILCALLS`, `DEFAULTDB`, `RUNQUERY`
- Updated: `WRITEBACK` only applies to main-memory document instances.
- Updated: `DEBUG` option can be changed at runtime by users with admin permissions.
- Updated: default of `INTPARSE` is now `false`.
- Removed: `HTMLOPT` (replaced with `HTMLPARSER`), `PARSEROPT` (replaced with parser-specific options), `DOTDISPLAY`, `DOTTY`

Version 7.7

- Added: `ADDCACHE`, `CHECKSTRINGS`, `FTINDEXSPLITSIZE`, `INDEXSPLITSIZE`

Version 7.6

- Added: `GLOBALLOCK`
- Added: store local options in configuration file after `# Local Options` comments.

Version 7.5

- Added: options can now be set via system properties

- Added: a pragma expression can be used to locally change database options
- Added: `USER`, `PASSWORD`, `LOG`, `LOGMSGMAXLEN`, `WEBPATH`, `RESTXQPATH` `HTTPLOCAL`, `CREATEONLY`, `STRIPNS`
- Removed: `HTTPPATH`; `HTTPPORT`: `jetty.xml` configuration file is used instead
- Removed: global options cannot be changed anymore during the lifetime of a BaseX instance

Version 7.3

- Updated: `KEEPALIVE`, `TIMEOUT`: default values changed
- Removed: `WILDCARDS`; new index supports both fuzzy and wildcard queries
- Removed: `SCORING`; new scoring model will focus on lengths of text nodes and match options

Version 7.2

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

Version 7.1

- Added: `ADDRAW`, `MAXLEN`, `MAXCATS`, `UPDINDEX`
- Updated: `BINDINGS`

Version 7.0

- Added: `SERVERHOST`, `KEEPALIVE`, `AUTOFLUSH`, `QUERYPATH`

Part IV. Integration

Chapter 17. Integrating oXygen

Read this entry online in the BaseX Wiki.

This tutorial is part of the Getting Started Section. It describes how to access BaseX from the oXygen XML Editor. 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

Preparations

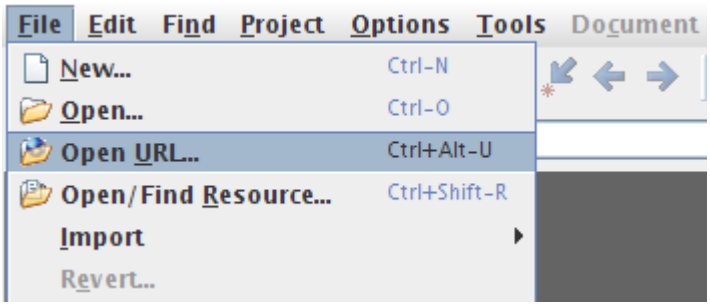
- First, start the BaseX WebDAV service.

Configuration

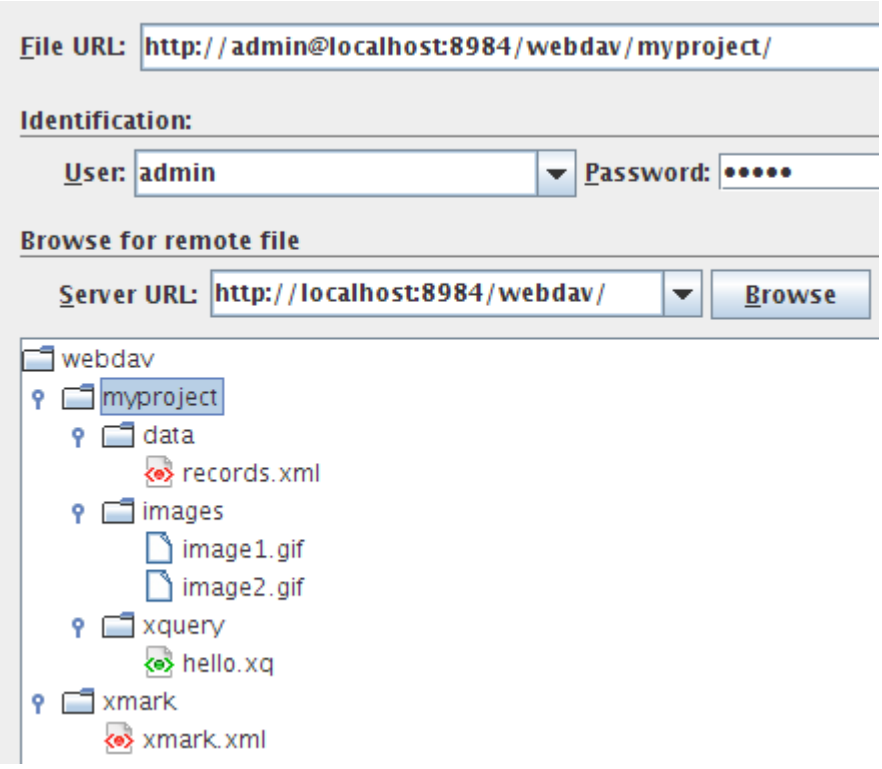
1. Go to menu *Options* → *Preferences* → *Data Sources*
2. In the Connections panel, click the *New* button (+)
3. Enter "BaseX-WebDAV" as connection name
4. Select "WebDAV" in the Data Source combo box
5. Fill in the appropriate connection details. Below, the default values are shown:
   - Set the URL to `http://localhost:8984/webdav`
   - Set the user name to `admin`
   - Set the password to `admin`
6. Now press *OK*, and your Data Source is ready for use

You can now open single database files as follows:

- 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 (ZIP, EXE)
2. Charles Foster's XQJ implementation provides a default (client/server) and a local driver. If you want to use the first flavor, you need to start a BaseX Server instance

Data Source

1. Start oXygen and go to *Options* → *Preferences* → *Data Sources*
2. Add a new Data Source with the *New* button (+)
3. Enter "BaseX-XQJ" as connection name and choose *XQuery API for Java (XQJ)* as type
4. Add the following JAR files above with the *Add* Button: `xqj-api-1.0.jar`, `xqj2-0.2.0.jar` and `basex-xqj-1.3.0.jar` (the versions of the JAR file may differ). If you add the BaseX library as well `basex7.9.jar`, you can also use the local XQJ driver.
5. Unter "Driver class", choose the default or the local XQJ driver (`net.xqj.basex.BaseXXQDataSource` vs. `net.xqj.basex.local.BaseXXQDataSource`).
6. Press *OK*

Connection

1. Press *New* (+) in the Connection Panel below.
2. Enter Name "BaseX" and select "BaseX-XQJ" in the Data Source box.
3. If you use the default driver, you need to enter connection details:
   - Port: `1984`

- Server name: `localhost`
- User: `admin`
- Password: `admin`

4. 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.

--CG 14:45, 29 January 2015 (CET)

Chapter 18. Integrating Eclipse

Read this entry online in the BaseX Wiki.

This article is part of the Getting Started Section. It describes how to run XPath/XQuery code from within the Eclipse IDE.

Another article describes how to compile and run BaseX with Eclipse.

Installation

The following steps apply to all operating systems:

- Install Version 3.7 (Indigo) of Eclipse: http://www.eclipse.org. Please note that more recent versions may work as well, but haven't been tested so far.
- download your favorite BaseX distribution (JAR, ZIP, EXE): http://basex.org/download/

Windows

It should be sufficient to install the official XQuery Development Tools Plugin (XQDT): http://www.xqdt.org/
Update Site: http://download.eclipse.org/webtools/incubator/repository/xquery/milestones/

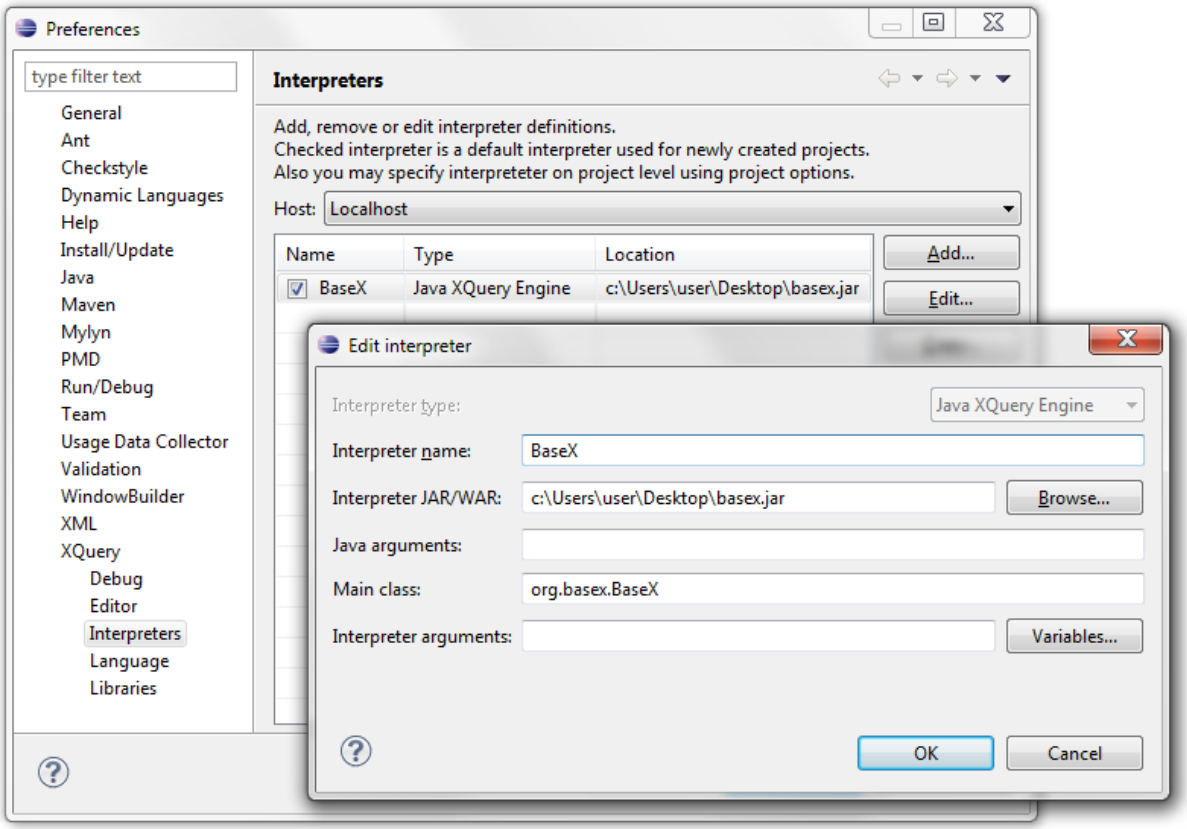
Linux

- First, install the Dynamic Languages Toolkit (DLTK)Update Site: http://download.eclipse.org/releases/indigo/
- Next, install Marklogic's XQDT Dropin

Mac OSX

- Install Marklogic's XQDT Dropin

Setting up



Use BaseX as query processor in Eclipse You can set up the XQuery interpreter as standalone or client version, as shown on the screenshot:

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.

Part V. Query Features

Chapter 19. XQuery

Read this entry online in the BaseX Wiki.

Welcome to the Query Portal, which is one of the Main Sections of this documentation. BaseX provides an implementation of the W3 XPath and XQuery 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 demonstrate. This section contains information on the query processor and its extensions:

XQuery 3.0 and XQuery 3.1

Features of the new XQuery Recommendations.

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.

Chapter 20. XQuery 3.0

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It summarizes the most interesting features of the XQuery 3.0 Recommendations. XQuery 3.0 is fully supported by BaseX.

Enhanced FLWOR Expressions

Most clauses of FLWOR expressions can now be specified in an arbitrary order: additional `let` and `for` clauses can be put after a `where` clause, and multiple `where`, `order by` and `group by` statements can be used. This means that many nested loops can now be rewritten to a single FLWOR expression.

Example:

```
for $country in db:open('factbook')//country
where $country/@population > 100000000
let $name := $country/name[1]
for $city in $country//city[population > 1000000]
group by $name
return <country name='{ $name }'>{ $city/name }</country>
```

A new `count` clause enhances the FLWOR expression with a variable that enumerates the iterated tuples.

```
for $n in (1 to 10)[. mod 2 = 1]
count $c
return <number count="{ $c }" number="{ $n }"/>
```

The `allowing empty` provides functionality similar to outer joins in SQL:

```
for $n allowing empty in ()
return 'empty? ' || empty($n)
```

Window clauses provide a rich set of variable declarations to process sub-sequences of iterated tuples. An example:

```
for tumbling window $w in (2, 4, 6, 8, 10, 12, 14)
    start at $s when fn:true()
    only end at $e when $e - $s eq 2
return <window>{ $w }</window>
```

More information on window clauses, and all other enhancements, can be found in the specification.

Simple Map Operator

The simple map operator `!` provides a compact notation for applying the results of a first to a second expression: the resulting items of the first expression are bound to the context item one by one, and the second expression is evaluated for each item. The map operator may be used as replacement for FLWOR expressions:

Example:

```
(: Simple map notation :)
(1 to 10) ! element node { . },
(: FLWOR notation :)
for $i in 1 to 10
return element node { $i }
```

A map operator is defined to be part of a path expression, which may now mix path and map operators. In contrast to the path operator, the results of the map operator will not be made duplicate-free and returned in document order.

Group By

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

XQuery:

```
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 contained in the XMark Benchmark Suite to use `group by`. The query partitions the customers based on their income.

Result:

```
<challenge>4731</challenge>
<na>12677</na>
<preferred>314</preferred>
<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>
```

Moreover, a value can be assigned to the grouping variable. This is shown in the following example:

XQuery:

```
let $data :=
  <xml>
    <person country='USA' name='John'/>
    <person country='USA' name='Jack'/>
    <person country='Germany' name='Johann'/>
  </xml>
```

```
for $person in $data/person
group by $country := $person/@country/string()
return element persons {
  attribute country { $country },
  $person/@name ! element name { data() }
}
```

Result:

```
<persons country="USA">
  <name>John</name>
  <name>Jack</name>
</persons>
<persons country="Germany">
  <name>Johann</name>
</persons>
```

Try/Catch

The try/catch construct can be used to handle errors at runtime:

Example:

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

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

Within the scope of the catch clause, a number of variables are implicitly declared, giving information about the error that occurred:

- `$err:code` error code
- `$err:description` : error message
- `$err:value` : value associated with the error (optional)
- `$err:module` : URI of the module where the error occurred
- `$err:line-number` : line number where the error occurred
- `$err:column-number` : column number where the error occurred
- `$err:additional` : error stack trace

Switch

The switch 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`

The expression to evaluate can correspond to multiple input values.

Example:

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

Result: `red red`

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 write 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 or 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 QNames

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

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

The syntax differed in older versions of the XQuery 3.0 specification, in which 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 named *pipe characters*) can be used to concatenate strings. This operator is a shortcut for the `fn:concat()` 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 namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:omit-xml-declaration "no";
declare option output:method "xhtml";
<html/>
```

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

In BaseX, the `output` prefix is statically bound and can thus be omitted. Note that all namespaces need to be specified when using external APIs, such as XQJ.

Context Item

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

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

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)
```

The following implementation-defined annotations are available:

- `%basex:inline([limit])` enforces the inlining of a function. Example:

Example:

```
declare option db:inlinelimit '0';
declare %basex:inline function local:id($x) { $x };
local:id(123)
```

In this query, function inlining has been deactivated by setting inlinelimit to `0`. The annotation enforces inlining for the given function, though, resulting in the optimized query expression `123`.

If an integer is specified as annotation argument, it will be interpreted a local inline limit.

- `%basex:lazy` enforces the lazy evaluation of a global variable. Example:

Example:

```
declare %basex:lazy variable $january := doc('does-not-exist');
if(month-from-date(current-date()) == 1) then $january else ()
```

The annotation ensures that an error will only be thrown if the condition yields true. Without the annotation, the error will always be thrown, because the referenced document is not found.

Functions

The following functions have been added in the XQuery 3.0 Functions and Operators Specification:

`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:parse-xml-fragment`, `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`, `fn:string-join`, `fn:node-name`, `fn:round`, `fn:data`

Changelog

Version 8.0

- Added: %basex:inline, %basex:lazy

Version 7.7

- Added: Enhanced FLWOR Expressions

Version 7.3

- Added: Simple Map Operator

Version 7.2

- Added: Annotations
- Updated: Expanded QNames

Version 7.1

- Added: Expanded QNames, Namespace Constructors

Version 7.0

- Added: String Concatenations

Chapter 21. Higher-Order Functions

Read this entry online in the BaseX Wiki.

This page talks about *higher-order functions* introduced with XQuery 3.0. The BaseX-specific Higher-Order Functions Module contains some additional useful functions.

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. This feature makes 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(
  $fun as function(item()) as item()*
) as function(item()*) as item()* {
  fn:for-each($fun, ?)
};
```

We'll see later how `fn:for-each(...)` 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:for-each` 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:for-each

Signatures	`fn:for-each($seq as item()*, $fun as function(item()) as item()*)) as item()*`
Summary	Applies the function item `$fun` 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:for-each(1 to 10, math:pow(?, 2))`
	Result: `1 4 9 16 25 36 49 64 81 100`
	• Applying a list of functions to a string:
	`let $fs := ( fn:upper-case#1, fn:substring(?, 4), fn:string-length#1 ) return fn:for-each($fs, function($f) { $f('foobar') })`
	Result: `FOOBAR bar 6`

XQuery 1.0	`declare function local:for-each( $seq as item()*, $fun as function(item()) as item()* ) as item()* { for $s in $seq return $fun($s) };`

fn:filter

Signatures	`fn:filter($seq as item()*, $pred as function(item()) as xs:boolean)) as item()*`
Summary	Applies the boolean predicate `$pred` to all elements of the sequence `$seq`, returning those for which it returns `true()`.
Examples	• All even integers until 10: `fn:filter(1 to 10, function($x) { $x mod 2 eq 0 })` *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(('FooBar', 'foo', 'BAR'), $first-upper)
```

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(1 to 20, $is-prime)
```

Result: `2 3 5 7 11 13 17 19`

Note

`fn:filter` can be easily implemented with `fn:for-each`:

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

XQuery 1.0

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

fn:for-each-pair

Signatures

`fn:for-each-pair($seq1 as item()*, $seq2 as item()*, $fun as function(item(), item()) 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:for-each-pair(
  fn:for-each(1 to 10, function($x) { $x mod 2 }),
  (1, 1, 1, 1, 1),
  function($a, $b) { $a + $b }
)
```

Result: `2 1 2 1 2`

- Line numbering:

```
let $number-lines := function($str) {
  fn:string-join(
    fn:for-each-pair(
```

```
      1 to 1000,
      tokenize($str, '\r?\n|\r'),
      concat(?, ': ', ?)
    ),
    '&#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:for-each-pair(
      $seq,
      fn:tail($seq),
      function($a, $b) { $a le $b }
    )
  satisfies $b
}
return (
  $is-sorted(1 to 10),
  $is-sorted((1, 2, 42, 4, 5))
)
```

Result: `true false`

XQuery 1.0

```
declare function local:for-each-pair(
  $seq1 as item()*,
  $seq2 as item()*,
  $fun as function(item(), item()) as item()*
) as item()* {
  for $pos in 1 to min((count($seq1), count($seq2)))
  return $fun($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 of 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

Signatures	`fn:fold-left($seq as item()*, $seed as item()*, $fun as function(item()*, item()) as item()*) as item()*`
Summary	The *left fold* traverses the sequence from the left. The query `fn:fold-left(1 to 5, 0, $f)` for example would be evaluated as: `$f($f($f($f($f(0, 1), 2), 3), 4), 5)`

Examples	• Product of a sequence of integers:
	`let $product := fn:fold-left(?, 1, function($result, $i) { $result * $i } ) return $product(1 to 5)`
	Result: `120`
	• Illustrating the evaluation order:
	`fn:fold-left(1 to 5, '$seed', concat('$f(', ?, ', ', ?, ')') )`
	Result: `$f($f($f($f($f($seed, 1), 2), 3), 4), 5)`
	• Building a decimal number from digits:
	`let $from-digits := fold-left(?, 0, function($n, $d) { 10 * $n + $d } ) return ( $from-digits(1 to 5), $from-digits((4, 2)) )`
	Result: `12345 42`

XQuery 1.0	As folds are more general than *FLWOR* expressions, the implementation isn't as concise as the former ones:

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

```
};
```

fn:fold-right

Signatures	`fn:fold-right($seq as item()*, $seed as item()*, $fun as function(item(), item()*) as item()*) as item()*`
Summary	The *right fold* `fn:fold-right($seq, $seed, $fun)` traverses the sequence from the right. The query `fn:fold-right(1 to 5, 0, $f)` for example would be evaluated as: `$f(1, $f(2, $f(3, $f(4, $f(5, 0)))))`

Examples	• Product of a sequence of integers:
	`let $product := fn:fold-right(?, 1, function($i, $result) { $result * $i } ) return $product(1 to 5)`
	Result: `120`
	• Illustrating the evaluation order:
	`fn:fold-right(1 to 5, '$seed', concat('$f(', ?, ', ', ?, ')') )`
	Result: `$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( $seq as item()*, $seed as item()*, $fun as function(item(), item()*) as item()* ) as item()* { if(empty($seq)) then $seed else $fun( fn:head($seq), local:fold-right(tail($seq), $seed, $fun) ) };` Note that the order of the arguments of `$fun` are inverted compared to that in `fn:fold-left(...)`.

Chapter 22. XQuery 3.1

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It summarizes the new features of the XQuery 3.1 Working Draft that are already supported by BaseX.

Maps

A *map* is a function that associates a set of keys with values, resulting in a collection of key/value pairs. Each key/value pair in a map is called an entry. A key is an arbitrary atomic value, and the associated value is an arbitrary sequence. Within a map, no two entries have the same key, when compared using the `eq` operator. It is not necessary that all the keys should be mutually comparable (for example, they can include a mixture of integers and strings).

Maps can be constructed as follows:

```
map { },                                        (: empty map :)
map { 'key': true(), 1984: (<a/>, <b/>) },  (: map with two entries :)
map:merge(                                      (: map with ten entries :)
  for $i in 1 to 10
  return map { $i: 'value' || $i }
)
```

The function corresponding to the map has the signature `function($key as xs:anyAtomicType) as item()*`. The expression `$map($key)` returns the associated value; the function call `map:get($map, $key)` is equivalent. For example, if `$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 $map := map { 'foo': 42, 'bar': 'baz', 123: 456 }
return fn:for-each(map:keys($map), $map)
```

This returns some permutation of `(42, 'baz', 456)`.

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`.

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.

Maps may be compared using the `fn:deep-equal` function. The Map Module describes the available set of map functions.

Arrays

An *array* is a function that associates a set of positions, represented as positive integer keys, with values. The first position in an array is associated with the integer `1`. The values of an array are called its members. In the type hierarchy, array has a distinct type, which is derived from function. In BaseX, arrays (as well as sequences) are based on an efficient Finger Tree implementation.

Arrays can be constructed in two ways. With the square bracket notation, the comma serves as delimiter:

```
[],             (: empty array :)
[ (1, 2) ],     (: array with single member :)
[ 1 to 2, 3 ]   (: array with two members; same as: [ (1, 2), 3 ] :)
```

With the `array` keyword and curly brackets, the inner expression is evaluated as usual, and the resulting values will be the members of the array:

```
array { },            (: empty array;               same as: array { () } :)
array { (1, 2) },     (: array with two members;    same as: array { 1, 2 } :)
array { 1 to 2, 3 }   (: array with three members; same as: array { 1, 2, 3 } :)
```

The function corresponding to the array has the signature `function($index as xs:integer) as item()*`. The expression `$array($index)` returns an addressed member of the array. The following query returns the five array members `48 49 50 51 52` as result:

```
let $array := array { 48 to 52 }
for $i in 1 to array:size($array)
return $array($i)
```

Like all other values, arrays are immutable. For example, the `array:reverse` function creates a new array containing a re-ordering of the members of an existing array, but the existing array is not changed by the operation. Like sequences, arrays have no identity. It is meaningful to compare the contents of two arrays, but there is no way of asking whether they are "the same array": two arrays with the same content are indistinguishable.

Atomization

If an array is *atomized*, all of its members will be atomized. As a result, an atomized item may now result in more than one item. Some examples:

```
fn:data([1 to 2])        (: returns the sequence 1, 2 :)
[ 'a', 'b', 'c' ] = 'b'  (: returns true :)
<a>{ [ 1, 2 ] }</a>      (: returns <a>1 2</a> :)
array { 1 to 2 } + 3     (: error: the left operand returns two items :)
```

Atomization also applies to function arguments. The following query returns 5, because the array will be atomized to a sequence of 5 integers:

```
let $f := function($x as xs:integer*) { count($x) }
return $f([1 to 5])
```

However, the next query returns 1, because the array is already of the general type `item()`, and no atomization will take place:

```
let $f := function($x as item()*) { count($x) }
return $f([1 to 5])
```

Arrays can be compared with the `fn:deep-equal` function. The Array Module describes the available set of array functions.

Lookup Operator

The lookup operator provides some syntactic sugar to access values of maps or array members at a specified position. It is introduced by the question mark (`?`) and followed by a specifier. The specifier can be:

1. A wildcard `*`,

2. The name of the key,

3. The integer offset, or

4. Any other parenthesized expression.

The following example demonstrates the four alternatives:

```
let $map := map { 'R': 'red', 'G': 'green', 'B': 'blue' }
return (
  $map?*          (: 1. returns all values; same as: map:keys($map) ! $map(.) :),
  $map?R          (: 2. returns the value associated with the key 'R'; same as:
 $map('R') :),
  $map?('G','B')  (: 3. returns the values associated with the key 'G' and 'B' :)
),

let $array := [ 'one', 'two', 'three' ]
return (
  $array?*         (: 1. returns all values; same as: (1 to array:size($array)) !
 $array(.) :),
  $array?1         (: 2. returns the first value; same as: $array(1) :),
  $array?(2 to 3)  (: 3. returns the second and third values; same as: (1 to 2) !
 $array(.) :)
)
```

The lookup operator can also be used without left operand. In this case, the context item will be used as input. This query returns `Akureyri`:

```
for $map in (
  map { 'name': 'Guðrún', 'city': 'Reykjavík' },
  map { 'name': 'Hildur', 'city': 'Akureyri' }
)
return $map[?name = 'Hildur'] ?city
```

Arrow Operator

The arrow operator applies a function to a value. The value is used as the first argument to the function. It is introduced with the characters `=>`, and it is followed by the function to be called. If `$v` is a value and `f()` is a function, then `$v=>f()` is equivalent to `f($v)`, and `$v=>f($j)` is equivalent to `f($v, $j)`. This is further illustrated by an example:

```
(: Returns 3 :)
count(('A', 'B', 'C')),
('A', 'B', 'C') => count(),
('A', 'B', 'C') => (function( $sequence) { count( $sequence)})(),

(: Returns W-E-L-C-O-M-E :)
string-join(tokenize(upper-case('w e l c o m e')), '-'),
'w e l c o m e' => upper-case() => tokenize() => string-join('-'),

(: Returns xfmdpnf :)
codepoints-to-string(
  for $i in string-to-codepoints('welcome')
  return $i + 1
),
(for $i in 'welcome' => string-to-codepoints()
 return $i + 1) => codepoints-to-string()
```

The syntax makes nested function calls more readable, as it is easy to see if parentheses are balanced.

Serialization

Two Serialization methods have been added to the Serialization spec:

Adaptive Serialization

In BaseX, `adaptive` is used as the new default serialization method. It provides a textual representation for all XDM types, including maps and arrays, functions, attributes, and namespaces. All items will be separated using the value of the `item-separator` parameter, or a newline if no value is specified:

```
<element id='id0'/>/@id,
map { 'key': 'value' },
true#0
```

Result:

```
 id="id0"
{
  "key": "value"
}
function true#0
```

JSON Serialization

The new `json` serialization output method can be used to serialize XQuery maps, arrays, atomic values and empty sequences as JSON.

The `json` output method has been introduced in BaseX quite a while ago. The implementation of this method now complies with the standard serialization rules and, at the same time, preserves the existing semantics:

- If an XML node of type `element(json)` is found, it will be serialized following the serialization rules of the JSON Module.
- Any other node or atomic value, map, array, or empty sequence will be serialized according to the rules in the specification.

The following two queries will both return the JSON snippet `{ "key": "value" }`:

```
declare option output:method 'json';
map { "key": "value" }
```

```
declare option output:method 'json';
<json type='object'>
  <key>value</key>
</json>
```

Functions

The following functions of the XQuery 3.1 Functions and Operators Working Draft have been added. Please be aware that the functions are still subject to change:

Map Functions

The following map functions are now available:

`map:merge`, `map:size`, `map:keys`, `map:contains`, `map:get`, `map:entry`, `map:put`, `map:remove`, `map:for-each`

Please check out the Map Module for more details.

Array Functions

The following array functions are now available:

`array:size`, `array:append`, `array:subarray`, `array:remove`, `array:insert-before`, `array:head`, `array:tail`, `array:reverse`, `array:join`, `array:flatten`, `array:for-each`, `array:filter`, `array:fold-left`, `array:fold-right`, `array:for-each-pair`

Please check out the Array Module for more details.

JSON Functions

XQuery now provides native support for JSON objects. Strings and resources can be parsed to XQuery items and, as shown above, serialized back to their original form.

fn:parse-json

Signatures

- `fn:parse-json($input as xs:string) as item()?`
- `fn:parse-json($input as xs:string, $options as map(*)) as item()?`

Parses the supplied string as JSON text and returns its item representation. The result may be a map, an array, a string, a double, a boolean, or an empty sequence. The allowed options can be looked up in the specification.

```
parse-json('{ "name": "john" }')   (: yields { "name": "json" } :),
parse-json('[ 1, 2, 4, 8, 16]')    (: yields [ 1, 2, 4, 8, 16 ] :)
```

fn:json-docs

Signatures

- `fn:json-doc($uri as xs:string) as item()?`
- `fn:json-doc($uri as xs:string, $options as map(*)) as item()?`

Retrieves the text from the specified URI, parses the supplied string as JSON text and returns its item representation (see fn:parse-json for more details).

```
json-doc("http://ip.jsontest.com/")('ip')  (: returns your IP address :)
```

fn:json-to-xml

Signatures

- `fn:json-to-xml($string as xs:string?) as node()?`

Converts a JSON string to an XML node representation. The allowed options can be looked up in the specification.

```
json-to-xml('{ "message": "world" }')

(: result:
<map xmlns="http://www.w3.org/2005/xpath-functions/json">
  <string key="message">world</string>
</map> :)
```

fn:xml-to-json

Signatures

- `fn:xml-to-json($node as node()?) as xs:string?`

Converts an XML node, whose format conforms to the results created by fn:json-to-xml, to a JSON string representation. The allowed options can be looked up in the specification.

```
(: returns "JSON" :)
xml-to-json(<string xmlns="http://www.w3.org/2005/xpath-functions/json">JSON</
string>)
```

fn:sort

Signatures

- `fn:sort($input as item()*) as item()*`
- `fn:sort($input as item()*, $key as function(item()*) as xs:anyAtomicType*)) as item()*`

Returns a new sequence with sorted `$input` items. If a sort `$key` function is given, it will be applied on all items. The items of the resulting values will be sorted using the semantics of the `lt` expression.

```
sort(reverse(1 to 3))                (: yields 1, 2, 3 :),
sort((3, -2, 1), abs#1)              (: yields 1, -2, 3 :),
sort((1,2,3), function($x) { -$x })  (: yields 3, 2, 1 :),
sort((1, 'a'))                       (: yields an error, as strings and integers
 cannot be compared :)
```

fn:contains-token

Signatures

- `fn:contains-token($input as xs:string*, $token as string) as xs:boolean`
- `fn:contains-token($input as xs:string*, $token as string, $collation as xs:string) as xs:boolean`

The supplied strings will be tokenized at whitespace boundaries. The function returns `true` if one of the strings equals the supplied token, possibly under the rules of a supplied collation:

```
contains-token(('a', 'b c', 'd'), 'c')               (: yields true :)
<xml class='one two'/>/contains-token(@class, 'one')  (: yields true :)
```

fn:parse-ietf-date

Signature

- `fn:parse-ietf-date($input as xs:string?) as xs:string?`

Parses a string in the IETF format (which is widely used on the Internet) and returns a `xs:dateTime` item:

```
fn:parse-ietf-date('28-Feb-1984 07:07:07')"             (: yields
 1984-02-28T07:07:07Z :),
fn:parse-ietf-date('Wed, 01 Jun 2001 23:45:54 +02:00')"  (: yields
 2001-06-01T23:45:54+02:00 :)
```

fn:apply

Signatures

- `fn:apply($function as function(*), $array as array(*)) as item()*`

A supplied function is invoked with the arguments supplied by an array. The arity of the function must be the same as the size of the array.

Example:

```
fn:apply(concat#5, array { 1 to 5 })            (: 12345 :)
fn:apply(function($a) { sum($a) }, [ 1 to 5 ])  (: 15 :)
fn:apply(count#1, [ 1,2 ])  (: error (the array has two members) :)
```

fn:random-number-generator

Signatures

- `fn:random-number-generator() as map(xs:string, item())`
- `fn:random-number-generator($seed as xs:anyAtomicType) as map(xs:string, item())`

Creates a random number generator, using an optional seed. The returned map contains three entries:

- `number` is a random double between 0 and 1
- `next` is a function that returns another random number generator
- `permute` is a function that returns a random permutation of its argument

The returned random generator is *deterministic*: If the function is called twice with the same arguments and in the same execution scope, it will always return the same result.

Example:

```
let $rng := fn:random-number-generator()
let $number := $rng('number')               (: returns a random number :)
let $next-rng := $rng('next')()             (: returns a new generator :)
let $next-number := $next-rng('number')     (: returns another random number :)
let $permutation := $rng('permute')(1 to 5) (: returns a random permutation of
 (1,2,3,4,5) :)
return ($number, $next-number, $permutation)
```

fn:format-number

The function has been extended to support scientific notation:

```
format-number(1984.42, '00.0e0')  (: yields 19.8e2 :)
```

fn:tokenize

If no separator is specified as second argument, a string will be tokenized at whitespace boundaries:

```
fn:tokenize("  a b  c d")  (: yields "a", "b", "c", "d" :)
```

fn:trace

The second argument can now be omitted:

```
fn:trace(<xml/>, "Node: ")/node()  (: yields the debugging output "Node: <xml/
>" :),
fn:trace(<xml/>)/node()            (: returns the debugging output "<xml/>" :)
```

Binary Data

Items of type `xs:hexBinary` and `xs:base64Binary` can now be compared against each other. The following queries all yield `true`:

```
xs:hexBinary('') < xs:hexBinary('bb'),
xs:hexBinary('aa') < xs:hexBinary('bb'),
max((xs:hexBinary('aa'), xs:hexBinary('bb'))) = xs:hexBinary('bb')
```

Collations

XQuery 3.1 provides a new default collation, which allows for a case-insensitive comparison of ASCII characters (`A-Z` = `a-z`). This query returns `true`:

```
declare default collation 'http://www.w3.org/2005/xpath-functions/collation/html-
ascii-case-insensitive';
'HTML' = 'html'
```

If the ICU Library is downloaded and added to the classpath, the full Unicode Collation Algorithm features get available in BaseX:

```
(: returns 0 (both strings are compared as equal) :)
compare('a-b', 'ab', 'http://www.w3.org/2013/collation/UCA?alternate=shifted')
```

Pending Features

The following functions have not been implemented yet:

- `fn:collation-key` , `fn:load-xquery-module`, `fn:transform`

Changelog

Version 8.2

- Added: fn:json-to-xml, fn:xml-to-json.

Version 8.1

- Updated: arrays are now based on an efficient Finger Tree implementation.

Introduced with Version 8.0.

Chapter 23. Module Library

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal.

In addition to the standard XQuery Functions, BaseX offers further function modules, which are listed in the following table. The module namespaces are statically bound, which means that they need not (but may) be explicitly declared in the query prolog.

Module	Description	Prefix	Namespace URI
Admin	Functions restricted to admin users.	`admin`	`http://basex.org/modules/admin`
Archive	Creating and processing ZIP archives.	`archive`	`http://basex.org/modules/archive`
Array	Functions for handling arrays.	`array`	`http://www.w3.org/2005/xpath-functions/array`
Binary	Processing binary data.	`bin`	`http://expath.org/ns/binary`
Client	Executing commands and queries on remote BaseX servers.	`client`	`http://basex.org/modules/client`
Conversion	Converting data (binary, numeric) to other formats.	`convert`	`http://basex.org/modules/convert`
Cryptography	Cryptographic functions, based on the EXPath Cryptograhic module.	`crypto`	`http://expath.org/ns/crypto`
CSV	Functions for processing CSV input.	`csv`	`http://basex.org/modules/csv`
Database	Functions for accessing and updating databases.	`db`	`http://basex.org/modules/db`
Fetch	Functions for fetching resources identified by URIs.	`fetch`	`http://basex.org/modules/fetch`
File	File handling, based on the latest draft of the EXPath File module.	`file`	`http://expath.org/ns/file`
Full-Text	Functions for performing full-text operations.	`ft`	`http://basex.org/modules/ft`
Hashing	Cryptographic hash functions.	`hash`	`http://basex.org/modules/hash`
Higher-Order	Additional higher-order functions that are not in the standard libraries.	`hof`	`http://basex.org/modules/hof`
HTML	Functions for converting HTML input to XML documents.	`html`	`http://basex.org/modules/html`

HTTP	Sending HTTP requests, based on the EXPath HTTP module.	`http`	`http://expath.org/ns/http-client`
Index	Functions for requesting details on database indexes.	`index`	`http://basex.org/modules/index`
Inspection	Functions for extracting internal module information.	`inspect`	`http://basex.org/modules/inspect`
JSON	Parsing and serializing JSON documents.	`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.	`math`	`http://www.w3.org/2005/xpath-functions/math`
Output	Functions for simplifying formatted output.	`out`	`http://basex.org/modules/out`
Process	Executing system commands from XQuery.	`proc`	`http://basex.org/modules/proc`
Profiling	Functions for profiling code snippets.	`prof`	`http://basex.org/modules/prof`
Random	Functions for creating random numbers.	`random`	`http://basex.org/modules/random`
Repository	Installing, deleting and listing packages.	`repo`	`http://basex.org/modules/repo`
SQL	JDBC bridge to access relational databases.	`sql`	`http://basex.org/modules/sql`
Streaming	Functions for handling streamable items.	`stream`	`http://basex.org/modules/stream`
Unit	Unit testing framework.	`unit`	`http://basex.org/modules/unit`
User	Creating and administering database users.	`user`	`http://basex.org/modules/user`
Validation	Validating documents against DTDs or XML Schema files.	`validate`	`http://basex.org/modules/validate`
Web	Convenience functions for building web applications.	`web`	`http://basex.org/modules/web`
XQuery	Evaluating new XQuery expressions at runtime.	`xquery`	`http://basex.org/modules/xquery`
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 module (soon obsolete).	`zip`	`http://expath.org/ns/zip`

For the following web application modules, the `basex-api` package must be included in the classpath and the modules must be imported in the query prolog. This is automatically the case if you use one of the complete distributions (zip, exe, war) of BaseX:

Module	Description	Prefix	Namespace URI
Geo	Functions for processing geospatial data.	`geo`	`http://expath.org/ns/geo`
Request	Server-side functions for handling HTTP Request data.	`request`	`http://exquery.org/ns/request`
RESTXQ	Helper functions for the RESTXQ API.	`rest`	`http://exquery.org/ns/restxq`
Session	Functions for handling server-side HTTP Sessions.	`session`	`http://basex.org/modules/session`
Sessions	Functions for managing all server-side HTTP Sessions.	`sessions`	`http://basex.org/modules/sessions`

Chapter 24. Repository

Read this entry online in the BaseX Wiki.

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.

Introduction

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. As XQuery comes with only 150 pre-defined functions, which cannot meet all requirements, there is some need for additional library modules – such as FunctX – that extend the language with new features.

BaseX offers the following mechanisms to make modules accessible to the XQuery processor:

1. The default Packaging mechanism will install single XQuery and Java modules in 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.

Accessing Modules

Library modules can be imported with the `import module` statement, followed by a freely choosable prefix and the namespace of the target module. The specified location may be absolute or relative; in the latter case, it is resolved against the location (i.e., *static base URI*) of the calling module. Import module statements must be placed at the beginning of a module:

Main Module `HelloUniverse.xq`:

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

Library Module `HelloWorld.xqm` (in the same directory):

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

Repository modules are stored in a directory named `BaseXRepo` or `repo`, which resides in your home directory. XQuery modules can be manually copied to the repository directory or installed and deleted via commands.

If a modules has been placed in the repository (see below), there is no need to specify the location. The following example calls a function from the FunctX module:

```
import module namespace functx = 'http://www.functx.com';
functx:capitalize-first('test')
```

Commands

There are various ways to handle your packages:

- Execute BaseX REPO commands (listed below)

- Call XQuery functions of the Repository Module

- Use the GUI (*Options* → *Packages*)

You can even manually add and remove packages in the repository directory; all changes will automatically be detected by the query processor.

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:

```
REPO INSTALL http://files.basex.org/modules/expath/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.

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 an additional argument, containing its name or 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

XQuery

If an XQuery file is specified as input for the install command, it will be parsed as XQuery library 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:

Installation (the original file will be copied to the `org/basex/modules/Hello.xqm` sub-directory of the repository):

```
REPO INSTALL http://files.basex.org/modules/org/basex/modules/Hello/HelloWorld.xqm
```

Importing the repository module:

```
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` 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 `Hello.java` (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` (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
org/basex/modules/Hello
org.basex.modules.Hello
```

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 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 XAR archive 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 offered by BaseX.

On our file server, 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:

1. If the URI is a URL…

   a. all colons will be replaced with slashes,

   b. in the URI authority, the order of all substrings separated by dots is reversed, and

   c. dots in the authority and the path are replaced by slashes. If no path exists, a single slash is appended.

2. *Version 8.2* : Otherwise, if the URI is a URN, all colons will be replaced with slashes.

3. If the resulting string ends with a slash, the `index` string is appended.

4. Characters other than letters, dots and slashes will be rewritten to hyphens.

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` → `a/little/example`
- `a:b:c` → `a/b/c`

Changelog

Version 8.2

- Added: URI Rewriting: support for URNs

Version 7.2.1

- Updated: Installation: existing packages will be replaced without raising an error
- Updated: Removal: remove specific version of a package
- Added: Packaging, URI Rewriting

Version 7.1

- Added: Repository Module

Version 7.0

- Added: EXPath Packaging

Chapter 25. Java Bindings

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It demonstrates two ways to invoke Java code from XQuery and 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. Variables are represented as function with 0 parameters.

The following example uses Java's `Math` class to return the cosine of an angle by calling the static method `cos()`, and the value of π by addressing the static variable via `PI()`:

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

The new Expanded QName notation of XQuery 3.0 allows you to directly specify a namespace URI instead of the prefix:

```
Q{java: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 called by passing on the resulting Java object as first argument.

In the following example, 256 bytes are written to the file `output.txt`. First, a new `FileWriter` instance is created, and its `write()` function is called in the next step. The `java:` prefix is omitted in the URI:

```
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)
)
```

Function names with dashes will be rewritten to Java's camel case notation:

```
XQuery: get-contents($x as xs:string)
Java  : getContents(String x)
```

Strings with invalid XML characters will be rejected by default. The validity check can be disabled by setting the CHECKSTRINGS option to false. The following query writes a file with a single 00-byte, which will then be successfully read via Java functions:

```
declare namespace br = 'java.io.BufferedReader';
declare namespace fr = 'java.io.FileReader';

declare option db:checkstrings 'false';
```

```
file:write-binary('00.bin', xs:hexBinary('00')),
br:new(fr:new('00.bin')) ! (br:readLine(.), br:close(.))
```

Note that Java code cannot be pre-compiled, and will often be evaluated slower than optimized XQuery code.

Module Imports

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 := (
  set:add("check"),
  set:add("what"),
  set:add("happens")
)
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. As a consequence, the addressed class must provide a constructor with no arguments.

Context-Awareness

Java classes can be coupled more closely to the BaseX core library. If a class inherits the abstract QueryModule class, the two variables queryContext and staticContext get available, which provide access to the global and static context of a 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 QueryResource interface can be implemented to enforce finalizing operations, such as the closing of opened connections or resources in a module. Its `close()` method will be called after a query has been fully evaluated.

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';

element user {
  context:user()
},
```

```
element to-int {
  try { context:to-int('abc') }
  catch * { 'Error in line', $err:line-number }
}
```

The imported Java class is shown below:

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

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

/**
 * This example inherits the {@link QueryModule} class and
 * implements the QueryResource interface.
 */
public class ContextModule extends QueryModule implements QueryResource {
  /**
   * Returns the name of the logged in user.
   * @return user
   */
  @Requires(Permission.NONE)
  @Deterministic
  @ContextDependent
  public String user() {
    return queryContext.context.user.name;
  }

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

  @Override
  public void close() {
    // see description above
  }
}
```

The result will look as follows:

```
<user>admin</admin>
<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.

Locking

By default, a Java function will be executed in parallel with other code. However, if a Java function performs sensitive write operations, it is advisable to explicitly lock the code. This can be realized via locking annotations:

```
@Lock(write = { "HEAVYIO" })
public void write() {
  // ...
}

@Lock(read = { "HEAVYIO" })
public void read() {
  // ...
}
```

If an XQuery expression is run which calls the Java `write()` function, every other query that calls `write()` or `read()` needs to wait for the query to be finished. If a query calls the `read()` function, only those queries are queued that call `write()`, because this function is only annotated with a `read` lock. More details on parallel query execution can be found in the article on Transaction Management.

Changelog

Version 8.0

- Added: `QueryResource` interface, called after a query has been fully evaluated.

Version 7.8

- Added: Java locking annotations
- Updated: `context` variable has been split into `queryContext` and `staticContext`.

Version 7.2.1

- Added: import of Java modules, context awareness

Chapter 26. Full-Text

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It summarizes the full-text and language-specific 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 Recommendation.

Introduction

The XQuery and XPath Full Text Recommendation (XQFT) is a feature-rich extension of the XQuery language. It can be used to both query XML documents and single strings for words and phrases. This section gives you a quick insight into the most important features of the language.

This is a simple example for a basic full-text expression:

```
"This is YOUR World" contains text "your world"
```

It yields `true`, because the search string is *tokenized* before it is compared with the tokenized input string. In the tokenization process, several normalizations take place. Many of those steps can hardly be simulated with plain XQuery: as an example, upper/lower case and diacritics (umlauts, accents, etc.) are removed and an optional, language-dependent stemming algorithm is applied. Beside that, special characters such as whitespaces and punctuation marks will be ignored. Thus, this query also yields true:

```
"Well... Done!" contains text "well, done"
```

The `occurs` keyword comes into play when more than one occurrence of a token is to be found:

```
"one and two and three" contains text "and" occurs at least 2 times
```

Varius range modifiers are available: `exactly`, `at least`, `at most`, and `from ... to ...`.

Combining Results

In the given example, curly braces are used to combine multiple keywords:

```
for $country in doc('factbook')//country
where $country//religions[text() contains text { 'Sunni', 'Shia' } any]
return $country/name
```

The query will output the names of all countries with a religion element containing `sunni` or `shia`. The `any` keyword is optional; it can be replaced with:

- `all` : all strings need to be found
- `any word` : any of the single words within the specified strings need to be found
- `all words` : all single words within the specified strings need to be found
- `phrase` : all strings need to be found as a single phrase

The keywords `ftand`, `ftor` and `ftnot` can also be used to combine multiple query terms. The following query yields the same result as the last one does (but it takes more memory):

```
doc('factbook')//country[descendant::religions contains text 'sunni' ftor 'shia']/
name
```

The keywords `not in` are special: they are used to find tokens which are not part of a longer token sequence:

```
for $text in ("New York", "new conditions")
return $text contains text "New" not in "New York"
```

Positional Filters

A popular retrieval operation is to filter texts by the distance of the searched words. In this query…

```
<xml>
  <text>There is some reason why ...</text>
  <text>For some good yet unknown reason, ...</text>
  <text>The reason why some people ...</text>
</xml>//text[. contains text { "some", "reason" } all ordered distance at most 3
 words]
```

…the two first texts will be returned as result, because there are at most three words between `some` and `reason`. Additionally, the `ordered` keyword ensures that the words are found in the specified order, which is why the third text is excluded. Note that `all` is required here to guarantee that only those hits will be accepted that contain all searched words.

The `window` keyword is related: it accepts those texts in which all keyword occur within the specified number of tokens. Can you guess what is returned by the following query?

```
("A C D", "A B C D E")[. contains text { "A", "E" } all window 3 words]
```

Sometimes it is interesting to only select texts in which all searched terms occur in the `same sentence` or `paragraph` (you can even filter for `different` sentences/paragraphs). This is obviously not the case in the following example:

```
'“I will survive!” This is what Mary told me.' contains text { 'will', 'told' } all
 words same sentence
```

Sentences are delimited by end of line markers (`.`, `!`, `?`, etc.), and newline characters are treated as paragraph delimiters. By the way: in the examples above, the `word` unit has been used, but `sentences` and `paragraphs` are valid alternatives.

Last but not least, three specifiers exist to filter results depending on the position of a hit:

- `at start` expects tokens to occur at the beginning of a text
- `at end` expects tokens to occur at the text end
- `entire content` only accepts texts which have no other words at the beginning or end

Match Options

As indicated in the introduction, the input and query texts are tokenized before they are compared with each other. During this process, texts are split into tokens, which are then normalized, based on the following matching options:

- If `case` is insensitive, no distinction is made between characters in upper and lower case. By default, the option is `insensitive`; it can also be set to `sensitive`:

```
"Respect Upper Case" contains text "Upper" using case sensitive
```

- If `diacritics` is insensitive, characters with and without diacritics (umlauts, characters with accents) are declared as identical. By default, the option is `insensitive`; it can also be set to `sensitive`:

```
"'Äpfel' will not be found..." contains text "Apfel" diacritics sensitive
```

- If `stemming` is activated, words are shortened to a base form by a language-specific stemmer:

```
"catch" contains text "catches" using stemming,
"Haus" contains text "Häuser" using stemming using language 'de'
```

- With the `stop words` option, a list of words can be defined that will be ignored when tokenizing a string. This is particularly helpful when the size of a full-text index structure needs to be reduced:

```
"You and me" contains text "you or me" using stop words ("and", "or"),
"You and me" contains text "you or me" using stop words at "http://files.basex.org/
etc/stopwords.txt"
```

- Related terms such as synonyms can be found with the sophisticated Thesaurus option.

The `wildcards` option facilitates search operations similar to simple regular expressions:

- `.` matches a single arbitrary character.
- `.?` matches either zero or one character.
- `.*` matches zero or more characters.
- `.+` matches one or more characters.
- `.{min,max}` matches *min–max* number of characters.

```
"This may be interesting in the year 2000" contains text { "interest.*", "2.
{3,3}" } using wildcards
```

This was a quick introduction to XQuery Full Text; you are invited to explore the numerous other features of the language!

BaseX Features

This page lists BaseX-specific full-text features and options.

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`).
- **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`).

- **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 : includes Snowball and Lucene stemmers and extends language support to the following languages: Bulgarian, Catalan, Czech, Danish, Dutch, Finnish, French, Greek, Hindi, Hungarian, Indonesian, Italian, Latvian, Lithuanian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish.

- igo-0.4.3.jar : 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.

The scoring model of BaseX takes into consideration the number of found terms, their frequency in a text, and the length of a text. The shorter the input text is, the higher scores will be:

```
(: Score values: 1 0.62 0.45 :)
for $text score $score in ("A", "A B", "A B C")[. contains text "A"]
order by $score descending
return <hit score='{ format-number($score, "0.00") }'>{ $text }</hit>
```

This simple approach has proven to consistently deliver good results, and in particular when little is known about the structure of the queried XML documents.

Please note that scores will only be computed if a parent expression requests them:

```
(: Computes and returns a scoring value. :)
let score $score := <x>Hello Universe</x> contains text "hello"
return $score

(: No scoring value will be computed here. :)
let $result := <x>Hello Universe</x> contains text "hello"
let score $score := $result
return $score
```

Scores will be propagated by the `and` and `or` expressions and in predicates. In the following query, all returned scores are equal:

```
let $text := "A B C"
let score $s1 := $text contains text "A" ftand "B C"
let score $s2 := $text contains text "A" ftand "B C"
let score $s3 := $text contains text "A" and $text contains text "B C"
let score $s4 := $text contains text "A" or $text contains text "B C"
```

```
let score $s5 := $text[. contains text "A"][. contains text "B C"]
return ($s1, $s2, $s3, $s4, $s5)
```

Thesaurus

BaseX supports full-text queries using thesauri, but it does not provide a default thesaurus. This is why queries 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. It is an XML with structure defined by an XSD Schema.

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 fulltext`

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 “house” and the text node “house”, and one error between “house” and “hous”.

Performance

Index Processing

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). 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.

FTAnd

The internal XQuery Full Text data model is pretty complex and may consume more main memory as would initially guess. If you plan to combine search terms via `ftand`, we recommend you to resort to an alternative, memory-saving representation:

```
(: representation via "ftand" :)
"A B" contains text "A" ftand "B" ftor "C" ftor "D"

(: memory saving representation :)
"A B" contains text { "A", "B" } all ftor { "C", "D" } all
```

Mixed Content

When working with so-called narrative XML documents, such as HTML, TEI, or DocBook documents, you typically have *mixed content*, i.e., elements containing a mix of text and markup, such as:

```
<p>This is only an illustrative <hi>example</hi>, not a <q>real</q> text.</p>
```

Since the logical flow of the text is not interrupted by the child elements, you will typically want to search across elements, so that the above paragraph would match a search for “real text”. For more examples, see XQuery and XPath Full Text 1.0 Use Cases.

To enable this kind of searches, *whitespace chopping* must be turned off when importing XML documents by setting the option `CHOP` to `OFF` (default: `SET CHOP ON`). In the GUI, you find this option in *Database* → *New…* → *Parsing* → *Chop Whitespaces*. A query such as `//p[. contains text 'real text']` will then match the example paragraph above. However, the full-text index will **not** be used in this query, so it may take a long time. The full-text index would be used for the query `//p[text() contains text 'real text']`, but this query will not find the example paragraph, because the matching text is split over two text nodes.

Note that the node structure is completely ignored by the full-text tokenizer: The `contains text` expression applies all full-text operations to the *string value* of its left operand. As a consequence, the `ft:mark` and `ft:extract` functions (see Full-Text Functions) will only yield useful results if they are applied to single text nodes, as the following example demonstrates:

```
(: Structure is ignored; no highlighting: :)
ft:mark(//p[. contains text 'real'])
(: Single text nodes are addressed: results will be highlighted: :)
ft:mark(//p[.//text() contains text 'real'])
```

Note that BaseX does **not** support the *ignore option* (`without content`) of the W3C XQuery Full Text 1.0 Recommendation. This means that it is not possible to ignore descendant element content, such as footnotes or other material that does not belong to the same logical text flow. Here is an example document:

```
<p>This text is provided for illustrative<note>Serving as an example or
 explanation.</note> purposes only.</p>
```

The ignore option would enable you to search for the string “illustrative purposes”:

```
//p[. contains text 'illustrative purposes' without content note]
```

For more examples, see XQuery and XPath Full Text 1.0 Use Cases.

As BaseX does not support the ignore option, it raises error FTST0007 when it encounters `without content` in a full-text `contains` expression.

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.

Collations

See XQuery 3.1 for standard collation features.

By default, string comparisons in XQuery are based on the Unicode codepoint order. The default namespace URI `http://www.w3.org/2003/05/xpath-functions/collation/codepoint` specifies this ordering. In BaseX, the following URI syntax is supported to specify collations:

```
 http://basex.org/collation?lang=...;strength=...;decomposition=...
```

Semicolons can be replaced with ampersands; for convenience, the URL can be reduced to its *query string component* (including the question mark). All arguments are optional:

Argument	Description
`lang`	A language code, selecting a Locale. It may be followed by a language variant. If no language is specified, the system’s default will be chosen. Examples: `de`, `en-US`.
`strength`	Level of difference considered significant in comparisons. Four strengths are supported: `primary`, `secondary`, `tertiary`, and `identical`. For example, in German, "Ä" and "A" are considered primary differences, "Ä" and "ä" are secondary differences, "Ä" and "Ä" are tertiary differences, and "A" and "A" are identical.
`decomposition`	Defines how composed characters are handled. Three decompositions are supported: `none`, `standard`, and `full`. More details are found in the JavaDoc of the JDK.

Some Examples:

- If a default collation is specified, it applies to all collation-dependent string operations in the query. The following expression yields `true`:

```
declare default collation 'http://basex.org/collation?lang=de;strength=secondary';
'Straße' = 'Strasse'
```

- Collations can also be specified in `order by` and `group by` clauses of FLWOR expressions. This query returns `à plutôt! bonjour!`:

```
for $w in ("bonjour!", "à plutôt!") order by $w collation "?lang=fr" return $w
```

- Various string function exists that take an optional collation as argument: The following functions give us `a` and `1 2 3` as results:

```
distinct-values(("a", "á", "à"), "?lang=it-IT;strength=primary"),
index-of(("a", "á", "à"), "a", "?lang=it-IT;strength=primary")
```

Changelog

Version 8.0

- Updated: Scores will be propagated by the `and` and `or` expressions and in predicates.

Version 7.7

- Added: Collations support.

Version 7.3

- Removed: Trie index, which was specialized on wildcard queries. The fuzzy index now supports both wildcard and fuzzy queries.
- Removed: TF/IDF scoring was discarded in favor of the internal scoring model.

Chapter 27. Full-Text: Japanese

Read this entry online in the BaseX Wiki.

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. Thank you to Toshio HIRAI for integrating the lexer in BaseX!

Introduction

The lexical analysis of Japanese documents is performed by Igo. 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 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 "##########(I wrote a book.)" is analyzed as follows.

```
##########
#       ##,###,##,*,*,*,#,###,###
#       ##,###,*,*,*,*,#,#,#
#       ##,##,*,*,*,*,#,##,##
#       ##,###,##,*,*,*,#,#,#
##     ##,##,*,*,########,###,##,##,##
##     ###,*,*,*,#####,###,##,##,##
#       ###,*,*,*,####,###,#,#,#
#       ##,##,*,*,*,*,#,#,#
```

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

```
##,#####1,#####2,#####3,###,###,##,##,##
(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, #, #, and ## will be passed to the indexer for each token.

Token Processing

"Fullwidth" and "Halfwidth" (which is defined by East Asian Width Properties) are not distinguished (this is the so-called ZENKAKU/HANKAKU problem). For example, ### and `XML` 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 or the Diacritics 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 "####### (I wrote the book)" and "###### (I write the book)" can be led back to the same prototype by analyzing their verb:

```
##    ##,##,*,*,########,###,[##],##,##

##    ##,##,*,*,########,#####,[##],##,##
#      ###,*,*,*,####,###,#,#,#
```

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:

```
'#######' contains text '##' using stemming using language 'ja'
'######' contains text '###' 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 '## ###(AKUTAGAWA, Ryunosuke)', a prominent Japanese writer, the first name of whom is often spelled as "## ##". The following two queries both return `true`:

```
'#####' contains text '.##' using wildcards using language 'ja'
'#####' contains text '.##' using wildcards using language 'ja'
```

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

```
'#####' contains text '##.##' 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:

```
'#####' contains text '##  .##' using wildcards using language 'ja'
```

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

```
'#####' contains text '##' ftand '.##' using wildcards using language 'ja'
```

Chapter 28. XQuery Update

Read this entry online in the BaseX Wiki.

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). 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.

Features

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>
  <title>Copy of: Transform expression example</title>
  <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.

Here is an example where we return an entire document, parts modified and all:

```
copy $c := doc("zaokeng.kml")
modify (
  for $d in $c//*:Point
  return insert node (
    <extrude>1</extrude>,
    <altitudeMode>relativeToGround</altitudeMode>
  )  before $d/*:coordinates
```

```
)
return $c
```

update

```
for $item in db:open('data')//item
return $item update delete node text()
```

The `update` expression is a convenience operator for writing simple transform expressions. Similar to the XQuery 3.0 map operator, the value of the first expression is bound as context item, and the second expression performs updates on this item. The updated item is returned as result.

Please note that `update` is not part of the official XQuery Update Facility yet. It is currently being discussed in the W3 Bug Tracker; your feedback is welcome.

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.

Database Functions

Some additional, updating database functions exist in order to perform updates on document and database level.

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. At the end of a query, after some consistency checks and optimizations, the update primitives will be applied in the following order:

- **Backups (1)** : `db:create-backup()`
- **XQuery Update** : `insert before`, `delete`, `replace`, `rename`, `replace value`, `insert attribute`, `insert into first`, `insert into`, `insert into last`, `insert`, `insert after`, `put`
- **Documents** : `db:add()`, `db:store()`, `db:replace()`, `db:rename()`, `db:delete()`, `db:optimize()`, `db:flush()`,
- **Users** : `user:grant()`, `user:password()`, `user:drop()`, `user:alter()`, `user:create()`
- **Databases** : `db:copy()`, `db:drop()`, `db:alter()`, `db:create()`
- **Backups (2)** : `db:restore()`, `db:drop-backup()`

If an inconsistency is found, an error message is returned and all accessed databases remain untouched (atomicity). For the user, this means that updates are only visible **after** the end of a snapshot.

It may be surprising to see `db:create` in the lower part of this list. This means that newly created database cannot be accessed by the same query, which can be explained by the semantics of updating queries: all expressions can

only be evaluated on databases that already exist while the query is evaluated. As a consequence, `db:create` is mainly useful in the context of Command Scripts, or Web Applications, in which a redirect to another page can be triggered after having created a database.

Example

The query…

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

…applied on the document…

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

…results in the following document:

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

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

Returning Results

By default, 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. But there are two ways out:

- The BaseX-specific `db:output()` function bridges this gap: it caches the results of its arguments at runtime and returns them after all updates have been processed. The following example performs an update and returns a success message:

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

- With the MIXUPDATES option, all updating constraints will be turned off. Returned nodes will be copied before they are modified by updating expressions. An error is raised if items are returned within a transform expression.

If you want to modify nodes in main memory, you can use the transform expression.

Function Declaration

To use updating expressions within a function, the `%updating` annotation 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

Original Files

In BaseX, all updates are performed on database nodes or in main memory. By default, update operations do not affect the original input file (the info string "Updates are not written back" appears in the query info to indicate this). The following solutions exist to write XML documents and binary resources to disk:

- Updates on main-memory instances of files that have been retrieved via `fn:doc` or `fn:collection` will be propagated back to disk when the `WRITEBACK` option is turned on. This option can also be activated on command line via `-u`. Make sure you back up the original documents before running your queries.
- Functions like `fn:put` or `file:write` can be used to write single XML documents to disk. With `file:write-binary`, you can write binary resources.
- The EXPORT command can be used write all resources of a databases to disk.

Indexes

Index structures are discarded after update operations when UPDINDEX is turned off (which is the default). More details are found in the article on Indexing.

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.

Changelog

Version 8.0

- Added: `MIXUPDATES` option for Returning Results in updating expressions
- Added: information message if files are not written back

Version 7.8

- Added: update convenience operator

Chapter 29. Serialization

Read this entry online in the BaseX Wiki.

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 3.1 document. In BaseX, they can be specified by…

- including them in the prolog of the XQuery expression,
- specifying them in the XQuery functions file:write() or fn:serialize(). The serialization parameters are specified as
  - children of an `<output:serialization-parameters/>` element, as defined for the fn:serialize() function, or as
  - map, which contains all key/value pairs: `map { "method": "xml", "cdata-section-elements": "div", ... }`,
- using the `-s` flag of the BaseX command-line clients,
- setting the SERIALIZER option before running a query,
- setting the EXPORTER option before exporting a database, or
- 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
`method`	Specifies the serialization method: • `xml` , `xhtml`, `html`, `text` and `adaptive` are adopted from the official specification. • `json` is specific to BaseX and can be used to output XML nodes as JSON objects (see the JSON Module for more details). • `csv` is BaseX-specific and can be used to output XML nodes as CSV data (see the CSV Module for more details). • `raw` is BaseX-specific, too: Binary data types are output in their *raw* form, i.e., without modifications. For all other types, the items' string values are returned. No indentation takes place, and and no characters are encoded via entities.	`adaptive`, `xml`, `xhtml`, `html`, `text`, `json`, `csv`, `raw`	`adaptive`
`version`	Specifies the version of the serialization method.	`xml/xhtml: 1.0, 1.1` `html: 4.0, 4.01, 5.0`	`1.0`
`html-version`	Specifies the version of the HTML serialization method.	`4.0, 4.01, 5.0`	`4.0`
`item-separator`	Determines a string to be used as item separator. If a separator is specified, the default separation of atomic values with single whitespaces will be skipped.	*arbitrary strings*, `\n`, `\r\n`, `\r`	*empty*

encoding	Encoding to be used for outputting the data.	*all encodings supported by Java*	UTF-8
indent	Adjusts whitespaces to make the output better readable.	yes, no	yes
cdata-section-elements	List of elements to be output as CDATA, separated by whitespaces.Example: <text><![CDATA[<>]]></text>		
omit-xml-declaration	Omits the XML declaration, which is serialized before the actual query resultExample: <?xml version="1.0" encoding="UTF-8"?>	yes, no	yes
standalone	Prints or omits the "standalone" attribute in the XML declaration.	yes, no, omit	omit
doctype-system	Introduces the output with a document type declaration and the given system identifier.Example: <!DOCTYPE x 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">		
undeclare-prefixes	Undeclares prefixes in XML 1.1.	yes, no	no
normalization-form	Specifies a normalization form. BaseX supports Form C (NFC).	NFC, none	NFC
media-type	Specifies the media type.		application/xml
parameter-document	Parses the value as XML document with additional serialization parameters (see the Serialization Specification for more details).		
use-character-maps	Defines character mappings. May only occur in documents parsed with parameter-document.		
byte-order-mark	Prints a byte-order-mark before starting serialization.	yes, no	no
escape-uri-attributes	Escapes URI information in certain HTML attributesExample: <a href="%C3%A4%C3%B6%C3%BC">äöü<a>	yes, no	no
include-content-type	Includes a meta content-type element if the result is output as HTMLExample: <head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></head>	yes, no	no

BaseX provides some additional serialization parameters:

Parameter	Description	Allowed	Default
csv	Defines the way how data is serialized as CSV.	see CSV Module	
json	Defines the way how data is serialized as JSON.	see JSON Module	
tabulator	Uses tab characters (\t) instead of spaces for indenting elements.	yes, no	no
indents	Specifies the number of characters to be indented.	*positive number*	2

`newline`	Specifies the type of newline to be used as end-of-line marker.	`\n`, `\r\n`, `\r`	*system dependent*
`limit`	Stops serialization after the specified number of bytes has been serialized. If a negative number is specified, everything will be output.	*positive number*	`-1`

The `csv` and `json` parameters are supplied with a list of options. Option names and values are combined with `=`, several options are separated by `,`:

Query:

```
(: The output namespace declaration is optional, because it is statically declared
 in BaseX) :)
declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:method "csv";
declare option output:csv "header=yes, separator=semicolon";
<csv>
  <record>
    <Name>John</Name>
    <City>Newton</City>
  </record>
  <record>
    <Name>Jack</Name>
    <City>Oldtown</City>
  </record>
</csv>
```

Result:

```
Name;City
John;Newton
Jack;Oldtown
```

Changelog

Version 8.0

- Added: Support for `use-character-maps` and `parameter-document`
- Added: Serialization method `adaptive`
- Updated: `adaptive` is new default method (before: `xml`)
- Removed: `format`, `wrap-prefix`, `wrap-uri`

Version 7.8.2

- Added: `limit`: Stops serialization after the specified number of bytes has been serialized

Version 7.8

- Added: `csv` and `json` serialization parameters
- Removed: `separator` option (use `item-separator` instead)

Version 7.7.2

- Added: `csv` serialization method
- Added: temporary serialization methods `csv-header`, `csv-separator`, `json-unescape`, `json-spec`, `json-format`

Version 7.5

- Added: official `item-separator` and `html-version` parameter
- Updated: `method=html5` removed; serializers updated with the latest version of the specification, using `method=html` and `version=5.0`.

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

Chapter 30. XQuery Errors

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It summarizes the codes of errors that are raised by the standard features and functions of XQuery. As the original specifications are pretty comprehensive, 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. Errors that are specific to BaseX can be found in the descriptions of the respective modules.

Original definitions of the error codes are found in the XQuery 3.0, XQuery 3.0 Functions, XQuery 1.0 Update, XQuery 1.0 Full Text, and EXPath HTTP Specifications.

Static Errors

Error Codes: `XPST`, `XQST`

Code	Description	Examples
`XPST0003`	An error occurred while *parsing* 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.	`1+ for i in //* return $i`
`XPST0005`	An expression will never return any results, no matter what input is provided.	`doc('input')/..`
`XPST0008`	A variable or type name is used that has not been defined in the current scope.	`$a--- element(*, x)`
`XPST0017`	• The specified function is unknown, or• it uses the wrong number of arguments.	`unknown() count(1,2,3)`
`XPST0051`	An unknown QName is used in a *sequence type* (e.g. in the target type of the `cast` expression).	`1 instance of x "test" cast as xs:itr`
`XPST0080`	`xs:NOTATION` or `xs:anyAtomicType` is used as target type of `cast` or `castable`.	`1 castable as xs:NOTATION`
`XPST0081`	• A QName uses a prefix that has not been bound to any namespace, or• a pragma or option declaration has not been prefixed.	`unknown:x (# pragma #) { 1 }`
`XQST0009`	The query imports a schema (schema import is not supported by BaseX).	`import schema "x"; ()`
`XQST0022`	Namespace values must be constant strings.	`<elem xmlns="{ 'dynamic' }"/>`
`XQST0031`	The specified XQuery version is not specified.	`xquery version "9.9"; ()`
`XQST0032`	The base URI was declared more than once.	`declare base-uri ...`
`XQST0033`	A namespace prefix was declared more than once.	`declare namespace a="a";declare namespace a="b"; ()`
`XQST0034`	A function was declared more than once.	`declare function local:a() { 1 };declare function local:a() { 2 }; local:a()`
`XQST0038`	The default collation was declared more than once.	`declare default collation ...`
`XQST0039`	Two or more parameters in a user-defined function have the same name.	`declare function local:fun($a, $a) { $a * $a };local:fun(1,2)`

XQDY0040	Two or more attributes in an element have the same node name.	`<elem a="1" a="12"/>`
XQDY0045	A user-defined function uses a reserved namespace.	`declare function fn:fun() { 1 }; ()`
XQST0047	A module was defined more than once.	`import module ...`
XQST0048	A module declaration does not match the namespace of the specified module.	`import module namespace invalid="uri"; 1`
XQST0049	A global variable was declared more than once.	`declare variable $a := 1;declare variable $a := 1; $a`
XQST0054	A global variable depends on itself. This may be triggered by a circular variable definition.	`declare variable $a := local:a();declare function local:a() { $a }; $a`
XQST0055	The mode for copying namespaces was declared more than once.	`declare copy-namespaces ...`
XQST0057	The namespace of a schema import may not be empty.	`import schema ""; ()`
XQST0059	The schema or module with the specified namespace cannot be found or processed.	`import module "unknown"; ()`
XQST0060	A user-defined function has no namespace.	`declare default function namespace "";declare function x() { 1 }; 1`
XQST0065	The ordering mode was declared more than once.	`declare ordering ...`
XQST0065	The default namespace mode for elements or functions was declared more than once.	`declare default element namespace ...`
XQST0067	The construction mode was declared more than once.	`declare construction ...`
XQST0068	The mode for handling boundary spaces was declared more than once.	`declare boundary-space ...`
XQST0069	The default order for empty sequences was declared more than once.	`declare default order empty ...`
XQST0070	A namespace declaration overwrites a reserved namespace.	`declare namespace xml=""; ()`
XQST0071	A namespace is declared more than once in an element constructor.	`<a xmlns="uri1" xmlns="uri2"/>`
XQST0075	The query contains a validate expression (validation is not supported by BaseX).	`validate strict { () }`
XQST0076	A `group by` or `order by` clause specifies an unknown collation.	`for $i in 1 to 10order by $i collation "unknown"return $i`
XQST0079	A pragma was specified without the expression that is to be evaluated.	`(# xml:a #) {}`
XQST0085	An empty namespace URI was specified.	`<pref:elem xmlns:pref=""/>`
XQST0087	An unknown encoding was specified. Note that the encoding declaration is currently ignored in BaseX.	`xquery version "1.0" encoding "a b"; ()`
XQST0088	An empty module namespace was specified.	`import module ""; ()`
XQST0089	Two variables in a `for` or `let` clause have the same name.	`for $a at $a in 1 return $i`
XQST0090	A character reference specifies an invalid character.	`"&#0;"`
XQST0093	A module depends on itself. This may be triggered by a circular module definition.	`import module ...`

XQST0094	`group by` references a variable that has not been declared before.	`for $a in 1 group by $b return $a`
XQST0097	A `decimal-format` property is invalid.	`declare default decimal-format digit = "xxx"; 1`
XQST0098	A single `decimal-format` character was assigned to multiple properties.	`declare default decimal-format digit = "%"; 1`
XQST0099	The context item was declared more than once.	`declare context item ...`
XQST0106	An annotation has been declared twice in a variable or function declaration.	`declare %updating %updating function ...`
XQST0108	Output declarations may only be specified in the main module.	Module: `declare output ...`
XQST0109	The specified serialization parameter is unknown.	`declare option output:unknown "..."; 1`
XQST0110	A serialization parameter was specified more than once in the output declarations.	`declare option output:indent "no";declare option output:indent "no"; 1`
XQST0111	A decimal format was declared more than once.	`declare decimal-format ...`
XQST0113	Context item values may only be in the main module.	Module: `declare context item := 1;`
XQST0114	A decimal-format property has been specified more than once.	`declare decimal-format EN NaN="!" NaN="?"; ()`

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).	`1 + "A"` `abs("a")` `1 cast as xs:gYear`
XPTY0018	The result of the last step in a path expression contains both nodes and atomic values.	`doc('input.xml')/(*, 1)`
XPTY0019	The result of a step (other than the last step) in a path expression contains an atomic values.	`(1 to 10)/*`
XQTY0024	An attribute node cannot be bound to its parent element, as other nodes of a different type were specified before.	`<elem>text { attribute a { "val" } }</elem>`
XQTY0105	A function item has been specified as content of an element.	`<X>{ false#0 }</X>`

Dynamic Errors

Error Codes: `XPDY`, `XQDY`

Code	Description	Examples
XPDY0002	• No value has been defined for an external variable, or• no context item has been set before the query was executed.	`declare variable $x external;` `$x descendant::*`
XPDY0050	• The operand type of a `treat` expression does not match the type of the argument, or• the root of the context item must be a document node.	`"string" treat as xs:int` `"string"[/]`

XQDY0025	Two or more attributes in a constructed element have the same node name.	`element x { attribute a { "" } attribute a { "" } }`
XQDY0026	The content of a computed processing instruction contains "?>".	`processing-instruction pi { "?>" }`
XQDY0041	The name of a processing instruction is invalid.	`processing-instruction { "1" } { "" }`
XQDY0044	The node name of an attribute uses reserved prefixes or namespaces.	`attribute xmlns { "etc" }`
XQDY0064	The name of a processing instruction equals "XML" (case insensitive).	`processing-instruction xml { "etc" }`
XQDY0072	The content of a computed comment contains "--" or ends with "-".	`comment { "one -- two" }`
XQDY0074	The name of a computed attribute or element is invalid, or uses an unbound prefix.	`element { "x y" } { "" }`
XQDY0095	A sequence with more than one item was bound to a `group by` clause.	`let $a := (1,2) group by $a return $a`
XQDY0096	The node name of an element uses reserved prefixes or namespaces.	`element { QName("uri", "xml:n") } {}`
XQDY0101	Invalid namespace declaration.	`namespace xmlns { 'x' }`
XQDY0102	Duplicate namespace declaration.	`element x { namespace a {'b'}, namespace a {'c'} }`

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.	`1 div 0`
FOAR0002	A numeric declaration or operation causes an over- or underflow.	`12345678901234567890` `xs:double("-INF") idiv 1`
FOCA0002	• A float number cannot be converted to a decimal or integer value, or• a function argument cannot be converted to a valid QName.	`xs:int(xs:double("INF"))` `QName("", "el em")`
FOCA0003	A value is too large to be represented as integer.	`xs:integer(99e100)`
FOCA0005	`"NaN"` is supplied to duration operations.	`xs:yearMonthDuration("P1Y") * xs:double("NaN")`
FOCH0001	A codepoint was specified that does not represent a valid XML character.	`codepoints-to-string(0)`
FOCH0002	A unsupported collation was specified in a function.	`compare('a', 'a', 'unknown')`
FOCH0003	A unsupported normalization form was specified in a function.	`normalize-unicode('a', 'unknown')`
FODC0001	The argument specified in `fn:id()` or `fn:idref()` must have a document node as root.	`id("id0", <xml/>)`
FODC0002	The specified document resource cannot be retrieved.	`doc("unknown.xml")`
FODC0004	The specified collection cannot be retrieved.	`collection("unknown")`
FODC0005	The specified URI to a document resource is invalid.	`doc("<xml/>")`
FODC0006	The string passed to `fn:parse-xml()` is not well-formed.	`parse-xml("<x/")`

FODC0007	The base URI passed to `fn:parse-xml()` is invalid.	`parse-xml("<x/>", ":")`
FODF1280	The name of the decimal format passed to `fn:format-number()` is invalid.	`format-number(1, "0", "invalid")`
FODF1310	The picture string passed to `fn:format-number()` is invalid.	`format-number(1, "invalid")`
FODT0001	An arithmetic duration operation causes an over- or underflow.	`xs:date('2000-01-01') + xs:duration('P99999Y')`
FODT0002	A duration declaration or operation causes an over- or underflow.	`implicit-timezone() div 0`
FODT0003	An invalid timezone was specified.	`adjust-time-to-timezone(xs:time("01:01:01"), xs:dayTimeDuration("PT20H"))`
FOER0000	Error triggered by the `fn:error()` function.	`error()`
FOFD1340	The picture string passed to `fn:format-date()`, `fn:format-time()` or `fn:format-dateTime()` is invalid.	`format-date(current-date(), "[]")`
FOFD1350	The picture string passed to `fn:format-date()`, `fn:format-time()` or `fn:format-dateTime()` specifies an non-available component.	`format-time(current-time(), "[Y2]")`
FONS0004	A function has a QName as argument that specifies an unbound prefix.	`resolve-QName("x:e", <e/>)`
FORG0001	A value cannot be cast to the required target type.	`xs:integer("A") 1 + <x>a</x>`
FORG0002	The URI passed to `fn:resolve-URI()` is invalid.	`resolve-URI(":")`
FORG0003	`fn:zero-or-one()` was called with more than one item.	`zero-or-one((1, 2))`
FORG0004	`fn:one-or-more()` was called with zero items.	`one-or-more(())`
FORG0005	`fn:exactly-one()` was called with zero or more than one item.	`exactly-one((1, 2))`
FORG0006	A wrong argument type was specified in a function call.	`sum((1, "string"))`
FORG0008	The arguments passed to `fn:dateTime()` have different timezones.	`dateTime(xs:date("2001-01-01+01:01"), current-time())`
FORX0001	A function specifies an invalid regular expression flag.	`matches('input', 'query', 'invalid')`
FORX0002	A function specifies an invalid regular expression.	`matches('input', '[')`
FORX0003	A regular expression matches an empty string.	`tokenize('input', '.?')`
FORX0004	The replacement string of a regular expression is invalid.	`replace("input", "match", "\")`
FOTY0012	An item has no typed value.	`count#1`
FOTY0013	Functions items cannot be atomized, have no defined equality, and have no string representation.	`data(false#0)`
FOTY0014	Function items have no string representation.	`string(map {})`
FOTY0015	Function items cannot be compared.	`deep-equal(false#0, true#0)`
FOUT1170	Function argument cannot be used to retrieve a text resource.	`unparsed-text(':')`

FOUT1190	Encoding to retrieve a text resource is invalid or not supported.	`unparsed-text('file.txt', 'InvalidEncoding')`

Serialization Errors

Error Codes: SEPM, SERE, SESU

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

Update Errors

Error Codes: FOUP, XUDY, XUST, XUTY

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

XUDY0023	The resulting update expression conflicts with existing namespaces.	`rename node <a:ns xmlns:a='uri'/> as QName('URI', 'a:ns')`
XUDY0024	New namespaces conflict with each other.	`copy $n := <x/> modify (insert node attribute { QName('uri1', 'a') } { "" } into $n, insert node attribute { QName('uri2', 'a') } { "" } into $n) return $n`
XUDY0027	Target of an update expression is an empty sequence.	`insert node <x/> into ()`
XUDY0029	The target of an update expression has no parent node.	`insert node <new/> before <target/>`
XUDY0030	Attributes cannot be inserted before or after the child of a document node.	`insert node <e a='a'/>/@a after document { <e/> }/*`
XUDY0031	Multiple calls to `fn:put()` address the same URI.	`for $i in 1 to 3 return put(<a/>, 'file.txt')`
XUST0001	No updating expression is allowed here.	`delete node /, "finished."`
XUST0002	An updating expression is expected in the `modify` clause or an updating function.	`copy $a := <x/> modify 1 return $a`
XUST0003	The revalidation mode was declared more than once.	`declare revalidation ...`
XUST0026	The query contains a revalidate expression (revalidation is not supported by BaseX).	`declare revalidation ...`
XUST0028	no return type may be specified in an updating function.	`declare updating function local:x() as item() { () }; ()`
XUTY0004	New attributes to be inserted must directly follow the root node.	`insert node (<a/>, attribute a {""}) into <a/>`
XUTY0005	A single element or document node is expected as target of an `insert` expression.	`insert node <new/> into attribute a { "" }`
XUTY0006	A single element, text, comment or processing instruction is expected as target of an `insert before/after` expression.	`insert node <new/> after attribute a { "" }`
XUTY0007	Only nodes can be deleted.	`delete node "string"`
XUTY0008	A single element, text, attribute, comment or processing instruction is expected as target of a `replace` expression.	`replace node document { <a/> } with <b/>`
XUTY0010	In a `replace` expression, in which no attributes are targeted, the replacing nodes must not be attributes as well.	`replace node <a><b/></a>/b with attribute size { 1 }`
XUTY0011	In the `replace` expression, in which attributes are targeted, the replacing nodes must be attributes as well.	`replace node <e a=""/>/@a with <a/>`
XUTY0012	In a `rename` expression, the target nodes must be an element, attribute or processing instruction.	`rename node text { 1 } as <x/>`
XUTY0013	An expression in the `copy` clause must return a single node.	`copy $c := (<a/>, <b/>) modify () return $c`
XUTY0022	An attribute must not be inserted into a document node.	`insert node <e a=""/>/@a into document {'a'}`

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 `not in` operator contains a *string exclude*.	`'a' contains text 'a' not in (ftnot 'a')`
FTDY0020	The search term uses an invalid wildcard syntax.	`'a' contains text '.{}' using wildcards`
FTST0007	The full-text expression contains an ignore option (the `ignore option` 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 specified more than once.	`'a' contains text 'a' using stemming using stemming`

BaseX Errors

Error Codes: BASX

Code	Description	Examples
BASX0000	Generic error, which is used for exceptions in context-aware Java bindings.	
BASX0001	The current user has insufficient permissions to execute an expression.	`file:delete('file.txt')`: *Create* rights needed.
BASX0002	The specified database option is unknown.	`declare option db:xyz "no"; 1`
BASX0003	Errors related to RESTXQ.	`%restxq:GET('x')`

Additional, module-specific error codes are listed in the descriptions of the query modules.

Part VI. XQuery Modules

Chapter 31. Admin Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for performing admin-centric operations such as managing database users and log data.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/admin` namespace, which is statically bound to the `admin` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

admin:sessions

Signatures	`admin:sessions() as element(session)*`
Summary	Returns an element sequence with all currently opened sessions, including the user name, address (IP:port) and an optionally opened database.The output of this function and the SHOW SESSIONS command is similar.
Examples	• `admin:sessions()` may e.g. return `<session user="admin" address="127.0.0.1:6286" database="factbook"/>`

admin:logs

Signatures	`admin:logs() as element(file)*` `admin:logs($date as xs:string) as element(entry)*` `admin:logs($date as xs:string, $merge as xs:boolean) as element(entry)*`
Summary	Returns Logging data compiled by the database or HTTP server: • If no argument is specified, a list of all log files will be returned, including the file size and date. • If a `$date` is specified, the contents of a single log file will be returned. • If `$merge` is set to true, related log entries will be merged. Please note that the merge might not be 100% successful, as log entries may be ambiguous.
Examples	• `admin:logs()` may return `<file size="834367"/>2015-01-23</file>` if a single log file exists. • `admin:logs() ! admin:logs(.)` lists the contents of all log files.

admin:write-log

Signatures	`admin:write-log($text as xs:string) as empty-sequence()`
Summary	Writes a string to the database logs, along with current user data (timestamp, user name). If the function is called in a web application or a database client, the IP will be logged. Otherwise, the string `STANDALONE` will be logged.

admin:delete-logs

Introduced with *Version 8.2*:

Signatures	`admin:delete-logs($date as xs:string) as empty-sequence()`

Summary	Deletes the log entries from the specified `$date`
Errors	`BXAD0001`: Today's log file cannot be deleted.`BXAD0002`: An error occurred while deleting a log file.

Errors

Code	Description
`BXAD0001`	Today's log file cannot be deleted.
`BXAD0002`	An error occurred while deleting a log file.

Changelog

Version 8.2

- Added: admin:delete-logs

Version 8.0

- Added: admin:write-log
- Deleted: admin:users (renamed to user:list-details)

Version 7.8.2

- Updated: admin:users: md5-encoded password added to output.
- Updated: admin:logs: represent name of log files as string value; `$merge` argument added.

The Module was introduced with Version 7.5.

Chapter 32. Archive Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to handle archives (including ePub, Open Office, JAR, and many other formats). New ZIP and GZIP archives can be created, existing archives can be updated, and the archive entries can be listed and extracted. The archive:extract-binary function includes an example for writing the contents of an archive to disk.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/archive` namespace, which is statically bound to the `archive` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

archive:create

Signatures	`archive:create($entries as item(), $contents as item()*) as xs:base64Binary` `archive:create($entries as item(), $contents as item()*, $options as item()) as xs:base64Binary`

Summary	Creates a new archive from the specified entries and contents.The `$entries` argument contains meta information required to create new entries. All items may either be of type `xs:string`, representing the entry name, or `element(archive:entry)`, containing the name as text node and additional, optional attributes:
	• `last-modified` : timestamp, specified as xs:dateTime (default: current time)
	• `compression-level` : 0-9, 0 = uncompressed (default: 8)
	• `encoding` : for textual entries (default: UTF-8)
	An example:
	`<archive:entry last-modified='2011-11-11T11:11:11' compression-level='8' encoding='US-ASCII'>hello.txt</archive:entry>`
	The actual `$contents` must be `xs:string` or `xs:base64Binary` items. The `$options` parameter contains archiving options, which can either be specified
	• as children of an `<archive:options/>` element:
	`<archive:options> <archive:format value="zip"/> <archive:algorithm value="deflate"/> </archive:options>`
	• as map, which contains all key/value pairs:
	`map { "format": "zip", "algorithm": "deflate" }`
	Currently, the following combinations are supported (all others will be rejected):
	• `zip` : `algorithm` may be `stored` or `deflate`

	• `gzip` : algorithm may be `deflate`
Errors	`ARCH0001`: the number of entries and contents differs.`ARCH0002`: the specified option or its value is invalid or not supported.`ARCH0003`: entry descriptors contain invalid entry names, timestamps or compression levels.`ARCH0004`: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.`ARCH0005`: the chosen archive format only allows single entries.`ARCH9999`: archive creation failed for some other reason.`FORG0006`: an argument has a wrong type.

Examples	The following one-liner creates an archive `archive.zip` with one file `file.txt`:
	
	`archive:create(<archive:entry>file.txt</archive:entry>, 'Hello World')`
	
	The following function creates an archive `mp3.zip`, which contains all MP3 files of a local directory:
	
	`let $path  := 'audio/'`
	`let $files := file:list($path, true(), '*.mp3')`
	`let $zip   := archive:create(`
	`  $files ! element archive:entry { . },`
	`  $files ! file:read-binary($path || .))`
	`return file:write-binary('mp3.zip', $zip)`

archive:entries

Signatures	`archive:entries($archive as xs:base64Binary) as element(archive:entry)*`

Summary	Returns the entry descriptors of the specified `$archive`. A descriptor contains the following attributes, provided that they are available in the archive format:
	
	• `size` : original file size
	
	• `last-modified` : timestamp, formatted as xs:dateTime
	
	• `compressed-size` : compressed file size
	
	An example:
	
	`<archive:entry size="1840" last-modified="2009-03-20T03:30:32"`
	` compressed-size="672">`
	`  doc/index.html`
	`</archive:entry>`

Errors	`ARCH9999`: archive creation failed for some other reason.
Examples	Sums up the file sizes of all entries of a JAR file: `sum(archive:entries(file:read-binary('zip.zip'))/@size)`

archive:options

Signatures	`archive:options($archive as xs:base64Binary) as element(archive:options)`
Summary	Returns the options of the specified `$archive` in the format specified by archive:create.
Errors	`ARCH0002`: The packing format is not supported.`ARCH9999`: archive creation failed for some other reason.
Examples	A standard ZIP archive will return the following options:

```
<archive:options xmlns:archive="http://basex.org/modules/archive">
  <archive:format value="zip"/>
  <archive:algorithm value="deflate"/>
</archive:options>
```

archive:extract-text

Signatures	`archive:extract-text($archive as xs:base64Binary) as xs:string*` `archive:extract-text($archive as xs:base64Binary, $entries as item()*) as xs:string*` `archive:extract-text($archive as xs:base64Binary, $entries as item()*, $encoding as xs:string) as xs:string*`
Summary	Extracts entries of the specified `$archive` and returns them as texts.The returned entries can be limited via `$entries`. The format of the argument is the same as for archive:create (attributes will be ignored).The encoding of the input files can be specified via `$encoding`.
Errors	`ARCH0004`: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.`ARCH9999`: archive creation failed for some other reason.
Examples	The following expression extracts all `.txt` files from an archive:

```
let $archive := file:read-binary("documents.zip")
for $entry in archive:entries($archive)[ends-with(., '.txt')]
return archive:extract-text($archive, $entry)
```

archive:extract-binary

Signatures	`archive:extract-binary($archive as xs:base64Binary) as xs:string*` `archive:extract-binary($archive as xs:base64Binary, $entries as item()*) as xs:base64Binary*`
Summary	Extracts entries of the specified `$archive` and returns them as binaries.The returned entries can be limited via `$entries`. The format of the argument is the same as for archive:create (attributes will be ignored).
Errors	`ARCH9999`: archive creation failed for some other reason.
Examples	This example unzips all files of an archive to the current directory:

```
let $archive  := file:read-binary('archive.zip')
let $entries  := archive:entries($archive)
let $contents := archive:extract-binary($archive)
for $entry at $p in $entries
return (
  file:create-dir(replace($entry, "[^/]+$", "")),
  file:write-binary($entry, $contents[$p])
)
```

archive:update

Signatures	`archive:update($archive as xs:base64Binary, $entries as item()*, $contents as item()*) as xs:base64Binary`
Summary	Creates an updated version of the specified `$archive` with new or replaced entries.The format of `$entries` and `$contents` is the same as for archive:create.
Errors	`ARCH0001`: the number of entries and contents differs.`ARCH0003`: entry descriptors contain invalid entry names, timestamps, compression levels or encodings.`ARCH0004`: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.`ARCH0005`: the entries of the given archive

	cannot be modified.`ARCH9999`: archive creation failed for some other reason.`FORG0006`: (some of) the contents are not of type `xs:string` or `xs:base64Binary`.
Examples	This example replaces texts in a Word document:

```
declare variable $input  := "HelloWorld.docx";
declare variable $output := "HelloUniverse.docx";
declare variable $doc    := "word/document.xml";

let $archive := file:read-binary($input)
let $entry   :=
  copy $c := fn:parse-xml(archive:extract-text($archive, $doc))
  modify replace value of node $c//*[text() = "HELLO WORLD!"] with
 "HELLO UNIVERSE!"
  return fn:serialize($c)
let $updated := archive:update($archive, $doc, $entry)
return file:write-binary($output, $updated)
```

archive:delete

Signatures	`archive:delete($archive as xs:base64Binary, $entries as item()*) as xs:base64Binary`
Summary	Deletes entries from an `$archive`.The format of `$entries` is the same as for archive:create.
Errors	`ARCH0005`: the entries of the given archive cannot be modified.`ARCH9999`: archive creation failed for some other reason.
Examples	This example deletes all HTML files in an archive and creates a new file:

```
let $zip := file:read-binary('old.zip')
let $entries := archive:entries($zip)[matches(., '\.x?html?$', 'i')]
return file:write-binary('new.zip', archive:delete($zip, $entries))
```

archive:write

Signatures	`archive:write($path as xs:string, $archive as xs:base64Binary) as xs:string* archive:write($path as xs:string, $archive as xs:base64Binary, $entries as item()*) as empty-sequence()`
Summary	This convenience function directly writes files of an `$archive` to the specified directory `$path`.The entries to be written can be limited via `$entries`. The format of the argument is the same as for archive:create (attributes will be ignored).
Errors	`FILE0001`: a specified path does not exist.`ARCH9999`: archive creation failed for some other reason.
Examples	This example unzips all files of an archive to the current directory:

```
archive:write('.', file:read-binary('archive.zip'))
```

Errors

Code	Description
`ARCH0001`	The number of specified entries and contents differs.
`ARCH0002`	The packing format or the specified option is invalid or not supported.
`ARCH0003`	Entry descriptors contain invalid entry names, timestamps or compression levels.
`ARCH0004`	The specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.

ARCH0005	The entries of the given archive cannot be modified.
ARCH0006	The chosen archive format only allows single entries.
ARCH9999	Archive processing failed for some other reason.

Changelog

Version 7.7

- Added: archive:write

The module was introduced with Version 7.3.

Chapter 33. Array Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for manipulating arrays, which will officially be introduced with XQuery 3.1. **Please note** that the functions are subject to change until the specification has reached its final stage.

Conventions

All functions in this module are assigned to the `http://www.w3.org/2005/xpath-functions/array` namespace, which is statically bound to the `array` prefix.

Functions

array:size

Signatures	`array:size($input as array(*)) as xs:integer`
Summary	Returns the number of members in `$array`. Note that because an array is an item, the `fn:count` function when applied to an array always returns `1`.
Examples	• `array:size([1 to 10])` returns `10`. • `array:size(array { 1 to 10 })` returns `10`.

array:get

Signatures	`array:get($array as array(*), $position as xs:integer) as item()*`
Summary	Returns the `$array` member at the specified `$position`.
Errors	`FOAY0001`: `$position` is not in the range `1` to `array:size($array)` inclusive.
Examples	• `array:get(array { reverse(1 to 5) }, 5)` returns the value `1`.

array:append

Signatures	`array:append($array as array(*), $member as item()*) as array(*)`
Summary	Returns a copy of `$array` with a new `$member` attached.
Examples	• `array:append([], 'member1')` returns the array `["member1"]`.

array:subarray

Signatures	`array:subarray($array as array(*), $position as xs:integer) as array(*)` `array:subarray($array as array(*), $position as xs:integer, $length as xs:integer) as array(*)`
Summary	Constructs a new array with with `$length` members of `$array` beginning from the specified `$position`.The two-argument version of the function returns the same result as the three-argument version when called with `$length` equal to the value of `array:size($array) - $position + 1`.
Errors	`FOAY0001`: `$position` is less than one, or if `$position + $length` is greater than `array:size($array) + 1`.`FOAY0002`: `$length` is less than zero.
Examples	• `array:append(['member1'], 'member2')` returns the array `["member1", "member2"]`.

array:remove

Signatures	`array:remove($array as array(*), $position as xs:integer) as array(*)`
Summary	Returns a copy of `$array` without the member at the specified `$position`.
Errors	`FOAY0001`: `$position` is not in the range `1` to `array:size($array)` inclusive.
Examples	• `array:append(["a"], 1)` returns the array `[]`.

array:insert-before

Signatures	`array:insert-before($array as array(*), $position as xs:integer, $member as item()*) as array(*)`
Summary	Returns a copy of `$array` with one new `$member` at the specified `$position`. Setting `$position` to the value `array:size($array) + 1` yields the same result as `array:append($array, $insert)`.
Errors	`FOAY0001`: `$position` is not in the range `1` to `array:size($array) + 1` inclusive.
Examples	• `array:insert-before(["a"], 1, "b")` returns the array `["b", "a"]`.

array:head

Signatures	`array:head($array as array(*)) as item()*`
Summary	Returns the first member of `$array`. This function is equivalent to the expression `$array(1)`.
Errors	`FOAY0001`: The array is empty.
Examples	• `array:head(["a", "b"])` returns `"a"`. • `array:head([["a", "b"], ["c", "d"]])` returns the array `["a", "b"]`.

array:tail

Signatures	`array:tail($array as array(*)) as array(*)`
Summary	Returns a new array with all members except the first from `$array`. This function is equivalent to the expression `array:remove($array, 1)`.
Errors	`FOAY0001`: The array is empty.
Examples	• `array:insert-before(["a"], 1, "b")` returns the array `["b", "a"]`.

array:reverse

Signatures	`array:reverse($array as array(*)) as array(*)`
Summary	Returns a new array with all members of `$array` in reverse order.
Examples	• `array:reverse(array { 1 to 3 })` returns the array `[3, 2, 1]`.

array:join

Signatures	`array:join($arrays as array(*)*) as array(*)`
Summary	Concatenates the contents of several `$arrays` into a single array.
Examples	• `array:join(())` returns the array `[]`. • `array:join((1 to 3) ! array { . })` returns the array `[1, 2, 3]`.

array:flatten

Signatures	`array:flatten($items as item()*) as item()*`
Summary	Recursively flattens all arrays that occur in the supplied `$items`.
Examples	• `array:flatten(["a","b"])` returns the sequence `"a", "b"`. • `array:flatten([1,[2,3],4]])` returns the sequence `1, 2, 3, 4`.

array:for-each

Signatures	`array:for-each($array as array(*), $function as function(item()*) as item()*) as array(*)`
Summary	Returns a new array, in which each member is computed by applying `$function` to the corresponding member of `$array`.

Examples	The following query returns the array `[2, 3, 4, 5, 6]`:
	`array:for-each(`
	`  array { 1 to 5 },`
	`  function($i) { $i + 1}`
	`)`

array:filter

Signatures	`array:filter($array as array(*), $function as function(item()*) as xs:boolean) as array(*)`
Summary	Returns a new array with those members of `$array` for which `$function` returns `true`.

Examples	The following query returns the array `[0, 1, 3]`:
	`array:filter(`
	`  array { 0, 1, -2, 3, -4 },`
	`  function($i) { $i > 0 }`
	`)`

array:fold-left

Signatures	`array:fold-left($array as array(*), $zero as item()*, $function as function(item()*, item()*) as item()*) as item()*`
Summary	Evaluates the supplied `$function` cumulatively on successive members of the supplied `$array` from left to right and using `$zero` as first argument.

Examples	The following query returns `55` (the sum of the integers 1 to 10):
	`array:fold-left(`
	`  array { 1 to 10 },`
	`  0,`
	`  function($a, $b) { $a + $b }`
	`)`

array:fold-right

Signatures	`array:fold-right($array as array(*), $zero as item()*, $function as function(item()*, item()*) as item()*) as item()*`
Summary	Evaluates the supplied `$function` cumulatively on successive members of the supplied `$array` from right to left and using `$zero` as first argument.

Examples	The following query is equivalent to the expression `array:reverse(array { 1 to 5 })`:

```
array {
  array:fold-right(
    array { 1 to 5 },
    (),
    function($a, $b) { $b, $a }
  )
}
```

array:for-each-pair

Signatures	`array:for-each-pair($array1 as array(*), $array2 as array(*), $function as function(item()*) as item()*) as array(*)`
Summary	Returns a new array obtained by evaluating the supplied `$function` for each pair of members at the same position in `$array1` and `$array2`.
Examples	The following query returns the array `[5, 7, 9]`:

```
array:for-each-pair(
  array { 1 to 3 },
  array { 4 to 6 },
  function($a + $b) { $a + $b }
)
```

array:sort

Signatures	`array:sort($array as array(*)) as array(*)` `array:sort($array as array(*), $key as function(item()*) as xs:anyAtomicType*) as array(*)`
Summary	Returns a new array with sorted `$array` members. If a sort `$key` function is given, it will be applied on all array members. The items of the resulting values will be sorted using the semantics of the `lt` expression.
Examples	• `array:sort(array { reverse(1 to 3) })` returns `[1, 2, 3]` • `array:sort([3, -2, 1], abs#1)` returns `[1, -2, 3]` • `array:sort([1,2,3], function($x) { -$x })` returns `[3, 2, 1]` • `array:sort((1, 'a'))` returns an error (strings and integers cannot be compared)

array:serialize

Signatures	`array:serialize($input as array(*)) as xs:string`
Summary	This function is specific to BaseX. It returns a string representation of the supplied array. The purpose of this function is to get an insight into the structure of an array item; it cannot necessarily be used for reconstructing the original array.
Examples	• `array:serialize([ 1, (2, 3), 4 to 6 ])` returns `[1, (2, 3), (4, 5, 6)]`.

Errors

Code	Description
`FOAY0001`	The specified index extends beyonds the bounds of an array.
`FOAY0002`	The specified length is less than zero.

Changelog

Introduced with Version 8.0.

Chapter 34. Binary Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to process binary data, including extracting subparts, searching, basic binary operations and conversion between binary and structured forms.

This module is based on the EXPath Binary Module.

Conventions

All functions and errors in this module are assigned to the `http://expath.org/ns/binary` namespace, which is statically bound to the `bin` prefix.

Constants and Conversions

bin:hex

Signatures	`bin:hex($in as xs:string?) as xs:base64Binary?`
Summary	Returns the binary form of the set of octets written as a sequence of (ASCII) hex digits ([0-9A-Fa-f]).`$in` will be effectively zero-padded from the left to generate an integral number of octets, i.e. an even number of hexadecimal digits. If `$in` is an empty string, then the result will be an `xs:base64Binary` with no embedded data. Byte order in the result follows (per-octet) character order in the string. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`non-numeric-character`: the input cannot be parsed as a hexadecimal number.
Examples	`bin:hex('11223F4E')` yields `ESI/Tg==`.`xs:hexBinary(bin:hex('FF'))` yields `FF`.

bin:bin

Signatures	`bin:bin($in as xs:string?) as xs:base64Binary?`
Summary	Returns the binary form of the set of octets written as a sequence of (8-wise) (ASCII) binary digits ([01]).`$in` will be effectively zero-padded from the left to generate an integral number of octets. If `$in` is an empty string, then the result will be an `xs:base64Binary` with no embedded data. Byte order in the result follows (per-octet) character order in the string. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`non-numeric-character`: the input cannot be parsed as a binary number.
Examples	`bin:bin('1101000111010101')` yields `0dU=`.`xs:hexBinary(bin:bin('1000111010101'))` yields `11D5`.

bin:octal

Signatures	`bin:octal($in as xs:string?) as xs:base64Binary?`
Summary	Returns the binary form of the set of octets written as a sequence of (ASCII) octal digits ([0-7]).`$in` will be effectively zero-padded from the left to generate an integral number of octets. If `$in` is an empty string, then the result will be an `xs:base64Binary` with no embedded data. Byte order in the result follows (per-octet) character order in the string. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`non-numeric-character`: the input cannot be parsed as an octal number.
Examples	`xs:hexBinary(bin:octal('11223047'))` yields `252627`.

bin:to-octets

Signatures	`bin:to-octets($in as xs:base64Binary) as xs:integer*`
Summary	Returns binary data as a sequence of octets.If `$in` is a zero length binary data then the empty sequence is returned. Octets are returned as integers from 0 to 255.

bin:from-octets

Signatures	`bin:from-octets($in as xs:integer*) as xs:base64Binary`
Summary	Converts a sequence of octets into binary data.Octets are integers from 0 to 255. If the value of `$in` is the empty sequence, the function returns zero-sized binary data.
Errors	`octet-out-of-range`: one of the octets lies outside the range 0 - 255.

Basic Operations

bin:length

Signatures	`bin:length($in as xs:base64Binary) as xs:integer`
Summary	Returns the size of binary data in octets.

bin:part

Signatures	`bin:part($in as xs:base64Binary?, $offset as xs:integer) as xs:base64Binary?` `bin:part($in as xs:base64Binary?, $offset as xs:integer, $size as xs:integer) as xs:base64Binary?`
Summary	Returns a section of binary data starting at the `$offset` octet.If `$size` is specified, the size of the returned binary data is `$size` octets. If `$size` is absent, all remaining data from `$offset` is returned. The `$offset` is zero based. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`negative-size`: the specified size is negative.`index-out-of-range`: the specified offset + size is out of range.
Examples	Test whether binary data starts with binary content consistent with a PDF file:`bin:part($data, 0, 4) eq bin:hex("25504446")`.

bin:join

Signatures	`bin:join($in as xs:base64Binary*) as xs:base64Binary`
Summary	Returns an `xs:base64Binary` created by concatenating the items in the sequence `$in`, in order. If the value of `$in` is the empty sequence, the function returns a binary item containing no data bytes.

bin:insert-before

Signatures	`bin:insert-before($in as xs:base64Binary?, $offset as xs:integer, $extra as xs:base64Binary?) as xs:base64Binary?`
Summary	Returns binary data consisting sequentially of the data from `$in` up to and including the `$offset - 1` octet, followed by all the data from `$extra`, and then the remaining data from `$in`.The `$offset` is zero based. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`index-out-of-range`: the specified offset is out of range.

bin:pad-left

Signatures	`bin:pad-left($in as xs:base64Binary?, $size as xs:integer) as xs:base64Binary?` `bin:pad-left($in as xs:base64Binary?, $size as xs:integer, $octet as xs:integer) as xs:base64Binary?`
Summary	Returns an `xs:base64Binary` created by padding the input with `$size` octets in front of the input. If `$octet` is specified, the padding octets each have that value, otherwise they are zero.If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`negative-size`: the specified size is negative.`octet-out-of-range`: the specified octet lies outside the range 0-255.

bin:pad-right

Signatures	`bin:pad-right($in as xs:base64Binary?, $size as xs:integer) as xs:base64Binary?` `bin:pad-right($in as xs:base64Binary?, $size as xs:integer, $octet as xs:integer) as xs:base64Binary?`
Summary	Returns an `xs:base64Binary` created by padding the input with `$size` octets after the input. If `$octet` is specified, the padding octets each have that value, otherwise they are zero.If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`negative-size`: the specified size is negative.`octet-out-of-range`: the specified octet lies outside the range 0-255.

bin:find

Signatures	`bin:find($in as xs:base64Binary?, $offset as xs:integer, $search as xs:base64Binary) as xs:integer?`
Summary	Returns the first location of the binary search sequence in the input, or if not found, the empty sequence.The `$offset` and the returned location are zero based. If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`index-out-of-range`: the specified offset + size is out of range.

Text Decoding and Encoding

bin:decode-string

Signatures	`bin:decode-string($in as xs:base64Binary?, $encoding as xs:string) as xs:string?` `bin:decode-string($in as xs:base64Binary?, $encoding as xs:string, $offset as xs:integer) as xs:string?` `bin:decode-string($in as xs:base64Binary?, $encoding as xs:string, $offset as xs:integer, $size as xs:integer) as xs:string?`
Summary	Decodes binary data as a string in a given `$encoding`.If `$offset` and `$size` are provided, the `$size` octets from `$offset` are decoded. If `$offset` alone is provided, octets from `$offset` to the end are decoded.If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`negative-size`: the specified size is negative.`index-out-of-range`: the specified offset + size is out of range.`unknown-encoding`: the specified encoding is unknown.`conversion-error`: an error or malformed input occurred during decoding the string.
Examples	Tests whether the binary data starts with binary content consistent with a PDF file:`bin:decode-string($data, 'UTF-8', 0, 4) eq '%PDF'`.

bin:encode-string

Signatures	`bin:encode-string($in as xs:string?, $encoding as xs:string) as xs:base64Binary?`

Summary	Encodes a string into binary data using a given `$encoding`.If the value of `$in` is the empty sequence, the function returns an empty sequence.
Errors	`unknown-encoding`: the specified encoding is unknown.`conversion-error`: an error or malformed input occurred during encoding the string.

Packing and Unpacking of Numeric Values

The functions have an optional parameter $octet-order whose string value controls the order: Least-significant-first order is indicated by any of the values `least-significant-first`, `little-endian`, or `LE`. Most-significant-first order is indicated by any of the values `most-significant-first`, `big-endian`, or `BE`.

bin:pack-double

Signatures	`bin:pack-double($in as xs:double) as xs:base64Binary` `bin:pack-double($in as xs:double, $octet-order as xs:string) as xs:base64Binary`
Summary	Returns the 8-octet binary representation of a double value.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified.
Errors	`unknown-significance-order`: the specified octet order is unknown.

bin:pack-float

Signatures	`bin:pack-float($in as xs:float) as xs:base64Binary` `bin:pack-float($in as xs:float, $octet-order as xs:string) as xs:base64Binary`
Summary	Returns the 4-octet binary representation of a float value.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified.
Errors	`unknown-significance-order`: the specified octet order is unknown.

bin:pack-integer

Signatures	`bin:pack-integer($in as xs:integer, $size as xs:integer) as xs:base64Binary` `bin:pack-integer($in as xs:integer, $size as xs:integer, $octet-order as xs:string) as xs:base64Binary`
Summary	Returns the twos-complement binary representation of an integer value treated as `$size` octets long. Any 'excess' high-order bits are discarded.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified. Specifying a `$size` of zero yields an empty binary data.
Errors	`unknown-significance-order`: the specified octet order is unknown.`negative-size`: the specified size is negative.

bin:unpack-double

Signatures	`bin:unpack-double($in as xs:base64Binary, $offset as xs:integer) as xs:double` `bin:unpack-double($in as xs:base64Binary, $offset as xs:integer, $octet-order as xs:string) as xs:double`
Summary	Extracts the double value stored at the particular offset in binary data.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified. The `$offset` is zero based.
Errors	`index-out-of-range`: the specified offset is out of range.`unknown-significance-order`: the specified octet order is unknown.

bin:unpack-float

Signatures	`bin:unpack-float($in as xs:base64Binary, $offset as xs:integer) as xs:float` `bin:unpack-float($in as xs:base64Binary, $offset as xs:integer, $octet-order as xs:string) as xs:float`
Summary	Extracts the float value stored at the particular offset in binary data.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified. The `$offset` is zero based.
Errors	`index-out-of-range`: the specified offset + size is out of range.`unknown-significance-order`: the specified octet order is unknown.

bin:unpack-integer

Signatures	`bin:unpack-integer($in as xs:base64Binary, $offset as xs:integer, $size as xs:integer) as xs:integer` `bin:unpack-integer($in as xs:base64Binary, $offset as xs:integer, $size as xs:integer, $octet-order as xs:string) as xs:integer`
Summary	Returns a signed integer value represented by the `$size` octets starting from `$offset` in the input binary representation. Necessary sign extension is performed (i.e. the result is negative if the high order bit is '1').Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified. The `$offset` is zero based. Specifying a `$size` of zero yields the integer `0`.
Errors	`negative-size`: the specified size is negative.`index-out-of-range`: the specified offset + size is out of range.`unknown-significance-order`: the specified octet order is unknown.

bin:unpack-unsigned-integer

Signatures	`bin:unpack-unsigned-integer($in as xs:base64Binary, $offset as xs:integer, $size as xs:integer) as xs:integer` `bin:unpack-unsigned-integer($in as xs:base64Binary, $offset as xs:integer, $size as xs:integer, $octet-order as xs:string) as xs:integer`
Summary	Returns an unsigned integer value represented by the `$size` octets starting from `$offset` in the input binary representation.Most-significant-octet-first number representation is assumed unless the `$octet-order` parameter is specified. The `$offset` is zero based. Specifying a `$size` of zero yields the integer `0`.
Errors	`negative-size`: the specified size is negative.`index-out-of-range`: the specified offset + size is out of range.`unknown-significance-order`: the specified octet order is unknown.

Bitwise Operations

bin:or

Signatures	`bin:or($a as xs:base64Binary?, $b as xs:base64Binary?) as xs:base64Binary?`
Summary	Returns the "bitwise or" of two binary arguments.If either argument is the empty sequence, an empty sequence is returned.
Errors	`differing-length-arguments`: the input arguments are of differing length.

bin:xor

Signatures	`bin:xor($a as xs:base64Binary?, $b as xs:base64Binary?) as xs:base64Binary?`

Summary	Returns the "bitwise xor" of two binary arguments.If either argument is the empty sequence, an empty sequence is returned.
Errors	`differing-length-arguments`: the input arguments are of differing length.

bin:and

Signatures	`bin:and($a as xs:base64Binary?, $b as xs:base64Binary?) as xs:base64Binary?`
Summary	Returns the "bitwise and" of two binary arguments.If either argument is the empty sequence, an empty sequence is returned.
Errors	`differing-length-arguments`: the input arguments are of differing length.

bin:not

Signatures	`bin:not($in as xs:base64Binary?) as xs:base64Binary?`
Summary	Returns the "bitwise not" of a binary argument.If the argument is the empty sequence, an empty sequence is returned.

bin:shift

Signatures	`bin:shift($in as xs:base64Binary?, $by as xs:integer) as xs:base64Binary?`
Summary	Shifts bits in binary data.If `$by` is zero, the result is identical to `$in`. If `$by` is positive then bits are shifted to the left. Otherwise, bits are shifted to the right. If the absolute value of `$by` is greater than the bit-length of `$in` then an all-zeros result is returned. The result always has the same size as `$in`. The shifting is logical: zeros are placed into discarded bits. If the value of `$in` is the empty sequence, the function returns an empty sequence.

Errors

Code	Description
`differing-length-arguments`	The arguments to a bitwise operation have different lengths.
`index-out-of-range`	An offset value is out of range.
`negative-size`	A size value is negative.
`octet-out-of-range`	An octet value lies outside the range 0-255.
`non-numeric-character`	Binary data cannot be parsed as number.
`unknown-encoding`	An encoding is not supported.
`conversion-error`	An error or malformed input during converting a string.
`unknown-significance-order`	An octet-order value is unknown.

Changelog

Introduced with Version 7.8.

Chapter 35. Client Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to access remote BaseX server instances from XQuery. With this module, you can on the one hand execute database commands and on the other hand evaluate queries, the results of which are returned as XDM sequences.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/client` namespace, which is statically bound to the `client` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

client:connect

Signatures	`client:connect($host as xs:string, $port as xs:integer, $user as xs:string, $password as xs:string) as xs:anyURI`
Summary	This function establishes a connection to a remote BaseX server, creates a new client session, and returns a session id. The parameter `$host` is the name of the database server, `$port` specifies the server port, and `$user` and `$password` represent the login data.
Errors	`BXCL0001`: an error occurs while creating the session (possible reasons: server not available, access denied).

client:execute

Signatures	`client:execute($id as xs:anyURI, $command as xs:string) as xs:string`
Summary	This function executes a command and returns the result as string. The parameter `$id` contains the session id returned by client:connect. The `$command` argument represents a single command, which will be executed by the server.
Errors	`BXCL0003`: an I/O error occurs while transferring data from or to the server.`BXCL0004`: an error occurs while executing a command.
Examples	The following query creates a new database `TEST` on a remote BaseX server:

```
client:connect('basex.server.org', 8080, 'admin', 'admin') !
  client:execute(., 'create database TEST')
```

client:info

Signatures	`client:info($id as xs:anyURI) as xs:string`
Summary	This function returns an information string, created by a previous call of client:execute. `$id` specifies the session id.

client:query

Signatures	`client:query($id as xs:anyURI, $query as xs:string) as item()*` `client:query($id as xs:anyURI, $query as xs:string, $bindings as map(*)) as item()*`
Summary	Evaluates a query and returns the result as sequence. The parameter `$id` contains the session id returned by client:connect, and `$query` represents the query string, which will be evaluated by the

	server.Variables and the context item can be declared via `$bindings`. The specified keys must be QNames or strings: • If a key is a QName, it will be directly adopted as variable name. • If a key is a string, it may be prefixed with a dollar sign. A namespace can be specified using the Clark Notation. If the specified string is empty, the value will be bound to the context item.
Errors	`BXCL0003`: an I/O error occurs while transferring data from or to the server.`BXCL0005`: an error occurs while evaluating a query, and if the original error cannot be extracted from the returned error string.`BXCL0006`: a value to be bound is no single item.

Examples	The following query sends a query on a local server instance, binds the integer `123` to the variable `$n` and returns `246`:
	`let $c := client:connect('localhost', 1984, 'admin', 'admin')`
	`return client:query($c, "declare variable $n external; $n * 2", map { 'n': 123 })`
	The following query performs a query on a first server, the results of which are passed on to a second server:
	`let $c1 := client:connect('basex1.server.org', 8080, 'jack', 'C0S19tt2X')`
	`let $c2 := client:connect('basex2.server.org', 8080, 'john', '465wFHe26')`
	`for $it in client:query($c1, '1 to 10')`
	`return client:query($c2, $it || '* 2')`

client:close

Signatures	`client:close($id as xs:anyURI) as empty-sequence()`
Summary	This function closes a client session. `$id` specifies the session id.At the end of query execution, open sessions will be automatically closed.
Errors	`BXCL0003`: an I/O error occurs while transferring data from or to the server.

Errors

Code	Description
`BXCL0001`	An error occurred while creating a new session (possible reasons: server not available, access denied).
`BXCL0002`	The specified session is unknown, or has already been closed.
`BXCL0003`	An I/O error occurred while transferring data from or to the server.
`BXCL0004`	An error occurred while executing a command.
`BXCL0005`	An error occurred while evaluating a query. Will only be raised if the XQuery error cannot be extracted from the returned error string.
`BXCL0006`	A value to be bound is no single item.

Changelog

Version 8.0

- Updated: Bound values may now contain no or more than one item in client:query.

Version 7.5

- Added: client:info

The module was introduced with Version 7.3.

Chapter 36. Conversion Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to convert data between different formats.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/convert` namespace, which is statically bound to the `convert` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Strings

convert:binary-to-string

Signatures	`convert:binary-to-string($bytes as xs:anyAtomicType) as xs:string` `convert:binary-to-string($bytes as xs:anyAtomicType, $encoding as xs:string) as xs:string`
Summary	Converts the specifed binary data (xs:base64Binary, xs:hexBinary) to a string.The UTF-8 default encoding can be overwritten with the optional `$encoding` argument.
Errors	`BXCO0001`: The input is an invalid XML string, or the wrong encoding has been specified. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.`BXCO0002`: The specified encoding is invalid or not supported.
Examples	• `convert:binary-to-string(xs:hexBinary('48656c6c6f576f726c64'))` returns the string `HelloWorld`.

convert:string-to-base64

Signatures	`convert:string-to-base64($input as xs:string) as xs:base64Binary` `convert:string-to-base64($input as xs:string, $encoding as xs:string) as xs:base64Binary`
Summary	Converts the specified string to a `xs:base64Binary` item. If the default encoding is chosen, conversion will be cheap, as both `xs:string` and `xs:base64Binary` items are internally represented as byte arrays.The UTF-8 default encoding can be overwritten with the optional `$encoding` argument.
Errors	`BXCO0001`: The input cannot be represented in the specified encoding.`BXCO0002`: The specified encoding is invalid or not supported.
Examples	• `convert:string-to-base64('HelloWorld')` returns the xs:base64binary item `SGVsbG9Xb3JsZA==`.

convert:string-to-hex

Signatures	`convert:string-to-hex($input as xs:string) as xs:hexBinary` `convert:string-to-hex($input as xs:string, $encoding as xs:string) as xs:hexBinary`
Summary	Converts the specified string to a `xs:hexBinary` item. If the default encoding is chosen, conversion will be cheap, as both `xs:string` and `xs:hexBinary` items are internally represented as byte arrays.The UTF-8 default encoding can be overwritten with the optional `$encoding` argument.
Errors	`BXCO0001`: The input cannot be represented in the specified encoding.`BXCO0002`: The specified encoding is invalid or not supported.

Examples	• `convert:string-to-hex('HelloWorld')` returns the Base64 item `48656C6C6F576F726C64`.

Binary Data

convert:bytes-to-base64

Signatures	`convert:bytes-to-base64($input as xs:byte*) as xs:base64Binary`
Summary	Converts the specified byte sequence to a `xs:base64Binary` item. Conversion is cheap, as `xs:base64Binary` items are internally represented as byte arrays.
Errors	`BXCO0001`: The input cannot be represented in the specified encoding.`BXCO0002`: The specified encoding is invalid or not supported.
Examples	• `convert:string-to-base64('HelloWorld')` returns the xs:base64binary item `SGVsbG9Xb3JsZA==`.

convert:bytes-to-hex

Signatures	`convert:bytes-to-hex($input as xs:byte*) as xs:hexBinary`
Summary	Converts the specified byte sequence to a `xs:hexBinary` item. Conversion is cheap, as `xs:hexBinary` items are internally represented as byte arrays.

convert:binary-to-bytes

Signatures	`convert:binary-to-bytes($bin as xs:anyAtomicType) as xs:byte*`
Summary	Returns the specified binary data (xs:base64Binary, xs:hexBinary) as a sequence of bytes.
Examples	• `convert:binary-to-bytes(xs:base64Binary('QmFzZVggaXMgY29vbA==']))` returns the sequence `(66, 97, 115, 101, 88, 32, 105, 115, 32, 99, 111, 111, 108)`. • `convert:binary-to-bytes(xs:hexBinary("4261736558"))` returns the sequence `(66 97 115 101 88)`.

Numbers

convert:integer-to-base

Signatures	`convert: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	• `convert:integer-to-base(-1, 16)` returns the hexadecimal string `'ffffffffffffffff'`. • `convert:integer-to-base(22, 5)` returns `'42'`.

convert:integer-from-base

Signatures	`convert: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 arbitarily.
Examples	• `convert:integer-from-base('ffffffffffffffff', 16)` returns `-1`. • `convert:integer-from-base('CAFEBABE', 16)` returns `3405691582`. • `convert:integer-from-base('42', 5)` returns `22`. • `convert:integer-from-base(convert:integer-to-base(123, 7), 7)` returns `123`.

Dates and Durations

convert:integer-to-dateTime

Signatures	`convert:integer-to-dateTime($ms as xs:integer) as xs:dateTime`
Summary	Converts the specified number of milliseconds since 1 Jan 1970 to an item of type xs:dateTime.
Examples	• `convert:integer-to-dateTime(0)` returns `1970-01-01T00:00:00Z`. • `convert:integer-to-dateTime(1234567890123)` returns `2009-02-13T23:31:30.123Z`.

convert:dateTime-to-integer

Signatures	`convert:dateTime-to-integer($dateTime as xs:dateTime) as xs:integer`
Summary	Converts the specified item of type xs:dateTime to the number of milliseconds since 1 Jan 1970.
Examples	• `convert:dateTime-to-integer(xs:dateTime('1970-01-01T00:00:00Z'))` returns `0`.

convert:integer-to-dayTime

Signatures	`convert:integer-to-dayTime($ms as xs:integer) as xs:dayTimeDuration`
Summary	Converts the specified number of milliseconds to an item of type xs:dayTimeDuration.
Examples	• `convert:integer-to-dayTime(1234)` returns `PT1.234S`.

convert:dayTime-to-integer

Signatures	`convert:dayTime-to-integer($dayTime as xs:dayTimeDuration) as xs:integer`
Summary	Converts the specified item of type xs:dayTimeDuration to milliseconds represented by an integer.
Examples	• `convert:dayTime-to-integer(xs:dayTimeDuration('PT1S'))` returns `1000`.

Errors

Code	Description
`BXCO0001`	The input is an invalid XML string, or the wrong encoding has been specified.
`BXCO0002`	The specified encoding is invalid or not supported.

Changelog

Version 7.5

- Added: `convert:integer-to-dateTime`, `convert:dateTime-to-integer`, `convert:integer-to-dayTime`, `convert:dayTime-to-integer`

The module was introduced with Version 7.3. Some of the functions have been adopted from the obsolete Utility Module.

Chapter 37. Cryptographic Module

Read this entry online in the BaseX Wiki.

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 and provides the following functionality: creation of message authentication codes (HMAC), encryption and decryption, and creation and validation of XML Digital Signatures.

Conventions

All functions in this module are assigned to the `http://expath.org/ns/crypto` namespace, which is statically bound to the `crypto` prefix. All errors are assigned to the `http://expath.org/ns/error` namespace, which is statically bound to the `experr` prefix.

Message Authentication

crypto:hmac

Signatures	`crypto:hmac($message as xs:string, $key as xs:string, $algorithm as xs:string) as xs:string` `crypto:hmac($message as xs:string, $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	`CX0013`: the specified hashing algorithm is not supported.`CX0014`: the specified encoding method is not supported.`CX0019`: 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 as xs:string, $key as xs:string, $algorithm as xs:string) as xs:string`
Summary	Encrypts the given input string. `$encryption` must be `symmetric`, as asymmetric encryption is not supported so far. **Default is `symmetric`** . `$key` is the secret key which is used for both encryption and decryption of input data. Its length is fixed and depends on the chosen algorithm: `8 bytes for DES`, `16 bytes for AES`. `$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	`CX0016`: padding problems arise.`CX0017`: padding is incorrect.`CX0018`: the encryption type is not supported.`CX0019`: the secret key is invalid.`CX0020`: the block size is incorrect.`CX0021`: 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, $type as xs:string, $key as xs:string, $algorithm as xs:string) as xs:string`
Summary	Decrypts the encrypted `$input`. `$type` must be `symmetric`. An option for asymmetric encryption will most likely be added with another version of BaseX. **Default is `symmetric`** . `$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`. `$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	`CX0016`: padding problems arise.`CX0017`: padding is incorrect.`CX0018`: the encryption type is not supported.`CX0019`: the secret key is invalid.`CX0020`: the block size is incorrect.`CX0021`: 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 Signatures

XML Signatures 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>
    <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 as node(), $canonicalization as xs:string, $digest as xs:string, $signature as xs:string, $prefix as xs:string, $type as xs:string) as node()` `crypto:generate-signature($input as node(), $canonicalization as xs:string, $digest as xs:string, $signature as xs:string, $prefix as xs:string, $type as xs:string, $xpath as xs:string, $certificate as node()) as node()` `crypto:generate-signature($input as node(), $canonicalization as xs:string, $digest as xs:string, $signature as xs:string, $prefix as xs:string, $type as xs:string, $ext as item()) as node()`
Summary	`$canonicalization` must either be `inclusive-with-comments`, `inclusive`, `exclusive-with-comments` or `exclusive`. **Default is `inclusive-with-comments`** . `$digest` must be one of the following: `SHA1`, `SHA256` or `SHA512`. **Default is `SHA1`** . `$signature` must either be `RSA_SHA1` or `DSA_SHA1`. **Default is `RSA_SHA1`** . `$prefix` may be empty and prefixes the `Signature` element accordingly. `$type` is the signature type. It must

	either be `enveloped` or `enveloping` (detached signatures are not supported so far). **Default is `enveloped`** . `$xpath` is an arbitrary XPath expression which specifies a subset of the document that is to be signed. `$certificate` is the digitial certificate used to sign the input document. `$ext` may either be an `$xpath` expression or a `$certificate`.
Errors	`CX0001`: the canonicalization algorithm is not supported.`CX0002`: the digest algorithm is not supported.`CX0003`: the signature algorithm is not supported.`CX0004`: the `$xpath-expression` is invalid.`CX0005`: the root name of `$digital-certificate` is not 'digital-certificate.`CX0007`: the key store is null.`CX0012`: the key cannot be found in the specified key store.`CX0023`: the certificate alias is invalid.`CX0024`: an invalid algorithm is specified.`CX0025`: an exception occurs while the signing the document.`CX0026`: an exception occurs during key store initialization.`CX0027`: an IO exception occurs.`CX0028`: the signature type is not supported.
Example	**Generates an XML Signature. 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	`CX0015`: the signature element cannot be found.`CX9994`: an unspecified problem occurs during validation.`CX9996`: an IO exception occurs during validation.
Example	**Validates an XML Signature. Query:**

```
let $sig := crypto:generate-signature(<a/>, '', '', '', '', '')
return crypto:validate-signature($sig)
```

Result:

```
true
```

Errors

Code	Description
CX0001	The canonicalization algorithm is not supported.
CX0002	The digest algorithm is not supported.
CX0003	The signature algorithm is not supported.
CX0004	The XPath expression is invalid.
CX0005	The root element of argument $digital-certificate must have the name 'digital-certificate'.
CX0006	The child element of argument $digital-certificate having position $position must have the name $child-element-name.
CX0007	The keystore is null.
CX0008	I/O error while reading keystore.
CX0009	Permission denied to read keystore.
CX0010	The keystore URL is invalid.
CX0011	The keystore type is not supported.
CX0012	Cannot find key for alias in given keystore.
CX0013	The hashing algorithm is not supported.
CX0014	The encoding method is not supported.
CX0015	Cannot find Signature element.
CX0016	No such padding.
CX0017	Incorrect padding.
CX0018	The encryption type is not supported.
CX0019	The secret key is invalid.
CX0020	Illegal block size.
CX0021	The algorithm is not supported.
CX0023	An invalid certificate alias is specified. Added to the official specification.
CX0024	The algorithm is invalid. Added to the official specification.
CX0025	Signature cannot be processed. Added to the official specification.
CX0026	Keystore cannot be processed. Added to the official specification.
CX0027	An I/O Exception occurred. Added to the official specification.
CX0028	The specified signature type is not supported. Added to the official specification.

Changelog

The Module was introduced with Version 7.0.

Chapter 38. CSV Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains a single function to parse CSV input. CSV (comma-separated values) is a popular representation for tabular data, exported e. g. from Excel.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/csv` namespace, which is statically bound to the `csv` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Conversion

XML: Direct, Attributes

CSV is converted to XML as follows:

- The resulting XML document has a `<csv>` root element.
- Rows are represented via `<record>` elements.
- Fields are represented via `<entry>` elements. The value of a field is represented as text node.
- If the `header` option is set to `true`, the first text line is parsed as table header, and the `<entry>` elements are replaced with the field names:
  - Empty names are represented by a single underscore (_), and characters that are not valid in element names are replaced with underscores or (when invalid as first character of an element name) prefixed with an underscore.
  - If the `lax` option is set to `false`, invalid characters will be rewritten to an underscore and the character's four-digit Unicode, and underscores will be represented as two underscores (__). The resulting element names may be less readable, but can always be converted back to the original field names.
- If `format` is set to `attributes`, field names will be stored in name attributes.

Map

If `format` is set to `map`, the CSV data will be converted to an XQuery map:

- All records are enumerated with positive integers.
- By default, all entries of a records are represented in a sequence.
- If the `header` option is set to `true`, a map is created, which contains all field names and its values.

A little advice: in the Database Creation dialog of the GUI, if you select CSV Parsing and switch to the *Parsing* tab, you can see the effects of some of the conversion options.

Options

The following options are available:

Option	Description	Allowed	Default
`separator`	Defines the character which separates the entries of a record in a single line.	`comma`, `semicolon`, `colon`, `tab`,	`comma`

		`space` or a *single character*	
`header`	Indicates if the first line of the parsed or serialized CSV data is a table header.	`yes`, `no`	`no`
`format`	Specifies the format of the XML data. The format is only relevant if the `header` option is activated: • With `direct` conversion, field names are represented as element names • With `attributes` conversion, field names are stored in `name` attributes • With `map` conversion, the input is converted to an XQuery map	`direct`, `attributes`, `map`	`direct`
`lax`	Specifies if a lax approach is used to convert QNames to JSON names.	`yes`, `no`	`yes`
`quotes`	Specifies if quotes should be parsed in the input and generated in the output.	`yes`, `no`	`yes`
`backslashes`	Specifies if characters are escaped by backslashes. Otherwise, a double quote is encoded by a second consecutive quote.	`yes`, `no`	`no`

The CSV function signatures provide an `$options` argument. Options can either be specified

- as children of an `<csv:options/>` element; e.g.:

```
<csv:options>
  <csv:separator value=';'/>
  ...
</csv:options>
```

- or as map, which contains all key/value pairs:

```
map { 'separator': ';', ... }
```

Functions

csv:parse

Signatures	`csv:parse($input as xs:string) as document-node(element(csv))` `csv:parse($input as xs:string, $options as item()) as item()`
Summary	Converts the CSV data specified by `$input` to an XML document or a map. The `$options` argument can be used to control the way the input is converted.
Errors	`BXCS0001`: the input cannot be parsed.

csv:serialize

Signatures	`csv:serialize($input as node()) as xs:string` `csv:serialize($input as node(), $options as item()) as xs:string`
Summary	Serializes the node specified by `$input` as CSV data, and returns the result as `xs:string`. Items can also be serialized as JSON if the Serialization Parameter `method` is set to `csv`.The `$options` argument can be used to control the way the input is serialized.
Errors	`BXCS0002`: the input cannot be serialized.

Examples

Example 1: Converts CSV data to XML, interpreting the first row as table header:

Input `addressbook.csv`:

```
Name,First Name,Address,City
Huber,Sepp,Hauptstraße 13,93547 Hintertupfing
```

Query:

```
let $text := file:read-text('addressbook.csv')
return csv:parse($text, map { 'header': true() })
```

Result:

```
<csv>
  <record>
    <Name>Huber</Name>
    <First_Name>Sepp</First_Name>
    <Address>Hauptstraße 13</Address>
    <City>93547 Hintertupfing</City>
  </record>
</csv>
```

Example 2: Converts some CSV data to XML and back, and checks if the input and output are equal. The expected result is `true`:

Query:

```
let $text := file:read-text('some-data.csv')
let $options := map { 'lax': false() }
let $xml := csv:parse($text, $options)
let $csv := csv:serialize($xml, $options)
return $text eq $csv
```

Example 3: Converts CSV data to an XQuery map item and serializes its contents:

Query:

```
let $text := "Name;City" || out:nl() || "John;Newton" || out:nl() || "Jack;Oldtown"
let $options :=
  <csv:options>
    <csv:separator value=';'/>
    <csv:format value='map'/>
    <csv:header value='yes'/>
  </csv:options>
return csv:parse($text, $options)
```

Result:

```
{
  1: {
    "City": "Newton",
    "Name": "John"
  },
  2: {
    "City": "Oldtown",
    "Name": "Jack"
```

```
  }
}
```

Errors

Code	Description
BXCS0001	The input cannot be parsed.
BXCS0002	The node cannot be serialized.

Changelog

Version 8.0

- Added: `backslashes` option

Version 7.8

- Updated: csv:parse now returns a document node instead of an element, or an XQuery map if `format` is set to `map`.
- Added: `format` and `lax` options

The module was introduced with Version 7.7.2.

Chapter 39. Database Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for processing databases from within XQuery. Existing databases can be opened and listed, its contents can be directly accessed, documents can be added to and removed, etc.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/db` namespace, which is statically bound to the `db` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Database Nodes

Database nodes are XML nodes which are either stored in a persistent database or part of a so-called *database fragment*. All XML fragments can be converted to database fragments by e. g. applying the transform expression on an XML fragment:

```
copy $c := element hello { 'world' } modify () return $c
```

General Functions

db:system

Signatures	`db:system() as element(system)`
Summary	Returns information on the database system, such as the database path and current database settings. The output is similar to the INFO command.

db:info

Signatures	`db:info($db as xs:string) as element(database)`
Summary	Returns meta information on the database `$db`. The output is similar to the INFO DB command.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.

db:list

Signatures	`db:list() as xs:string*` `db:list($db as xs:string) as xs:string*` `db:list($db as xs:string, $path as xs:string) as xs:string*`
Summary	Returns a string sequence with the names of all databases: • If a database `$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.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
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($db as xs:string) as element(resource)*` `db:list-details($db as xs:string, $path as xs:string) as element(resource)*`

Summary	• If no argument is specified, a sequence of elements is returned. A single element contains the name of a database, the number of stored resources, the date of modification, and the database path. • If `$db` is specified, a sequence of elements is returned, comprising information on all resources of the addressed database. An element contains the name of the resource, the content type, the modified date, and the raw flag. • Returned databases resources can be further restricted by the `$path` argument.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Examples	• `db:list-details("docs")` returns the names plus additional data of all documents from the database named `docs`.

db:backups

Signatures	`db:backups() as element(backup)*` `db:backups($db as xs:string) as element(backup)*`
Summary	Returns an element sequence containing all available database backups.If a database `$db` is specified, the sequence will be restricted to the backups matching this database.
Examples	• `db:backups("factbook")` returns all backups that have been made from the `factbook` database.

Read Operations

db:open

Signatures	`db:open($db as xs:string) as document-node()*` `db:open($db as xs:string, $path as xs:string) as document-node()*`
Summary	Opens the database `$db` and returns all document nodes.The document nodes to be returned can be filtered with the `$path` argument.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Examples	• `db:open("docs")` returns all documents from the database named `docs`. • `db:open("db", "one")` returns all documents from the database named `db` located in the path `one`. • `for $i in 1 to 3 return db:open("db" || $i)//item` returns all item elements from the databases `db1`, `db2` and `db3`.

db:open-pre

Signatures	`db:open-pre($db as xs:string, $pre as xs:integer) as node()`
Summary	Opens the database `$db` and returns the node with the specified `$pre` value.The PRE value provides very fast access to an existing database node, but it will change whenever a node with a smaller *pre* values is added to or deleted from a database.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0009`: 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:open-id

Signatures	`db:open-id($db as xs:string, $id as xs:integer) as node()`

Summary	Opens the database `$db` and returns the node with the specified `$id` value.Each database node has a *persistent* ID value. Access to the node id can be sped up by turning on the UPDINDEX option.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0009`: the specified id value does not exist in the database.

db:node-pre

Signatures	`db:node-pre($nodes as node()*) as xs:integer*`
Summary	Returns the *pre* values of the nodes supplied by `$nodes`, which must all be database nodes.The PRE value provides very fast access to an existing database node, but it will change whenever a node with a smaller *pre* values is added to or deleted from a database.
Errors	`BXDB0001`: `$nodes` contains a node which is not stored in a database.
Examples	• `db:node-pre(doc("input"))` returns `0` if the database `input` contains a single document.

db:node-id

Signatures	`db:node-id($nodes as node()*) as xs:integer*`
Summary	Returns the *id* values of the nodes supplied by `$nodes`, which must all be database nodes.Each database node has a *persistent* ID value. Access to the node id can be sped up by turning on the UPDINDEX option.
Errors	`BXDB0001`: `$nodes` contains a node which is not stored in a database.

db:retrieve

Signatures	`db:retrieve($db as xs:string, $path as xs:string) as xs:base64Binary`
Summary	Returns a binary resource addressed by the database `$db` and `$path` as streamable `xs:base64Binary`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0003`: the database is not *persistent* (stored on disk).`FODC0002`: the addressed resource cannot be retrieved.`FODC0007`: the specified path is invalid.
Examples	• `declare option output:method 'raw';db:retrieve("DB", "music/01.mp3")` returns the specified audio file as raw data. • `stream:materialize(db:retrieve("DB", "music/01.mp3"))` returns a materialized representation of the streamable result.

db:export

Signatures	`db:export($db as xs:string, $path as xs:string) as empty-sequence()` `db:export($db as xs:string, $path as xs:string, $params as item()) as empty-sequence()`
Summary	Exports the specified database `$db` to the specified file `$path`. Existing files will be overwritten. The `$params` argument contains serialization parameters (see Serialization for more details), which can either be specified • as children of an `<output:serialization-parameters/>` element, as defined for the fn:serialize() function; e.g.:

```
<output:serialization-parameters>
  <output:method value='xml'/>
  <output:cdata-section-elements value="div"/>
  ...
```

```
</output:serialization-parameters>
```

- as map, which contains all key/value pairs:

```
map { "method": "xml", "cdata-section-elements": "div", ... }
```

Errors	`BXDB0002`: The addressed database does not exist or could not be opened.

Examples	Export all files as text:
	`db:export("DB", "/home/john/xml/texts", map { 'method': 'text' })`
	The following query can be used to export parts of the database:
	`let $target := '/home/john/xml/target'`
	`for $doc in db:open('DB', 'collection')`
	`let $path := $target \|\| db:path($doc)`
	`return (`
	`  file:create-dir(file:parent($path)),`
	`  file:write($path, $doc)`
	`)`

Contents

db:text

Signatures	`db:text($db as xs:string, $string as item()) as text()*`
Summary	Returns all text nodes of the database `$db` that have `$string` as their string value. If available, the value index is used to speed up evaluation.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Examples	• `db:text("DB", "QUERY")/..` returns the parents of all text nodes of the database `DB` that match the string `QUERY`.

db:text-range

Signatures	`db:text-range($db as xs:string, $min as xs:string, $max as xs:string) as text()*`
Summary	Returns all text nodes of the database `$db` that are located in between the `$min` and `$max` strings. If available, the value index is used to speed up evaluation.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Examples	• `db:text-range("DB", "2000", "2001")` returns all text nodes of the database `DB` that are found in between `2000` and `2001`.

db:attribute

Signatures	`db:attribute($db as xs:string, $string as item()) as attribute()*` `db:attribute($db as xs:string, $string as item(), $attname as xs:string) as attribute()*`
Summary	Returns all attribute nodes of the database `$db` 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.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
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

Signatures	`db:attribute-range($db as xs:string, $min as xs:string, $max as xs:string) as attribute()*` `db:attribute-range($db as xs:string, $min as xs:string, $max as xs:string, $attname as xs:string) as attribute()*`
Summary	Returns all attributes of the database `$db`, 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.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
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`.

Updates

Important note: All functions in this section are *updating functions*: they will not be immediately executed, but queued on the Pending Update List, which will be processed after the actual query has been evaluated. This means that the order in which the functions are specified in the query does usually not reflect the order in which the code will be evaluated.

db:create

Signatures

`db:create($db as xs:string) as empty-sequence()` `db:create($db as xs:string, $inputs as item()*) as empty-sequence()` `db:create($db as xs:string, $inputs as item()*, $paths as xs:string*) as empty-sequence()` `db:create($db as xs:string, $inputs as item()*, $paths as xs:string*, $options as item()) as empty-sequence()`

Summary

Creates a new database with name `$db` and adds initial documents specified via `$inputs` to the specified `$paths`. An existing database will be overwritten.`$inputs` may be strings or nodes different than attributes. If the `$input` source is not a file or a folder, the `$paths` argument is mandatory.Please note that `db:create` will be placed last on the Pending Update List. As a consequence, a newly created database cannot be addressed in the same query.The `$options` argument can be used to change the indexing behavior. Allowed options are all parsing, XML parsing, indexing and full-text options in lower case. Options can be specified either...

- as children of an `<options/>` element, e.g.

```
<options>
  <textindex value='true'/>
  <maxcats value='128'/>
</options>
```

- or as map, which contains all key/value pairs:

```
map { "textindex": true(), "maxcats": 128 }
```

Errors

`FODC0002`: `$inputs` points to an unknown resource.`FOUP0001`: `$inputs` is neither string nor a document node.`BXDB0007`: `$db` is opened by another process.`BXDB0011`: `$db` is not a valid database name.`BXDB0012`: two `db:create` statements with the same database name were specified.`BXDB0013`: the number of specified inputs and paths differs.

Examples

- `db:create("DB")` creates the empty database `DB`.
- `db:create("DB", "/home/dir/doc.xml")` creates the database `DB` and adds the document `/home/dir/doc.xml` as initial content.
- `db:create("DB", <a/>, "doc.xml")` creates the database `DB` and adds the document with content `<a/>` under the name `doc.xml`.

- `db:create("DB", "/home/dir/", "docs/dir")` creates the database `DB` and adds the documents in `/home/dir` to the database under the path `docs/dir`.
- `db:create("DB", file:list('.'), map { 'ftindex': true() })` adds all files of the current working directory to a new database and creates a full-text index.

db:drop

Signatures	`db:drop($db as xs:string) as empty-sequence()`
Summary	Drops the database `$db` and all connected resources.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0007`: `$db` is opened by another process.
Examples	• `db:drop("DB")` drops the database `DB`.

db:add

Signatures	`db:add($db as xs:string, $input as item()) as empty-sequence()` `db:add($db as xs:string, $input as item(), $path as xs:string) as empty-sequence()` `db:add($db as xs:string, $input as item(), $path as xs:string, $options as item()) as empty-sequence()`
Summary	Adds documents specified by `$input` to the database `$db` and the specified `$path`. A document with the same path may occur more than once in a database. If this is unwanted, db:replace can be used.`$input` may be a string or a node different than attribute. If the `$input` source is not a file or a folder, `$path` must be specified.The `$options` argument can be used to control parsing. The syntax is identical to the db:create function. Allowed options are all parsing and XML parsing options.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`FODC0002`: `$input` points to an unknown resource.`FOUP0001`: `$input` is neither string nor 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 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 xs:string, $path as xs:string) as empty-sequence()`
Summary	Deletes document(s), specified by `$path`, from the database `$db`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
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:copy

Signatures	`db:copy($db as xs:string, $newname as xs:string) as empty-sequence()`
Summary	Creates a copy of the database specified by `$db` to `$newname`.

Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0011`: Invalid database name.`BXDB0016`: Name of source and target database is equal.

db:alter

Signatures	`db:alter($db as xs:string, $newname as xs:string) as empty-sequence()`
Summary	Renames the database specified by `$db` to `$newname`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0011`: Invalid database name.`BXDB0016`: Name of source and target database is equal.

db:create-backup

Signatures	`db:create-backup($db as xs:string) as empty-sequence()`
Summary	Creates a backup of the database `$db`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0011`: Invalid database name.
Examples	• `db:create-backup("DB")` creates a backup of the database `DB`.

db:drop-backup

Signatures	`db:drop-backup($name as xs:string) as empty-sequence()`
Summary	Drops all backups of the database with the specified `$name`. If the given `$name` points to a specific backup file, only this specific backup file is deleted.
Errors	`BXDB0002`: No backup file found.`BXDB0011`: Invalid database name.
Examples	• `db:drop-backup("DB")` drops all backups of the database `DB`. • `db:drop-backup("DB-2014-03-13-17-36-44")` drops the specific backup file `DB-2014-03-13-17-36-44.zip` of the database `DB`.

db:restore

Signatures	`db:restore($name as xs:string) as empty-sequence()`
Summary	Restores the database with the specified `$name`. The `$name` may include the timestamp of the backup file.
Errors	`BXDB0011`: Invalid database name.`BXDB0015`: No backup found.
Examples	• `db:restore("DB")` restores the database `DB`. • `db:restore("DB-2014-03-13-18-05-45")` restores the database `DB` from the backup file with the given timestamp.

db:optimize

Signatures	`db:optimize($db as xs:string) as empty-sequence()` `db:optimize($db as xs:string, $all as xs:boolean) as empty-sequence()` `db:optimize($db as xs:string, $all as xs:boolean, $options as item()) as empty-sequence()`
Summary	Optimizes the meta data and indexes of the database `$db`.If `$all` is `true`, the complete database will be rebuilt.The `$options` argument can be used to control indexing. The syntax is identical to the db:create function: Allowed options are all indexing and full-text options. UPDINDEX is only allowed if `$all` is `true`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`FOUP0002`: an error occurred while optimizing the database.

Examples	• `db:optimize("DB")` optimizes the database structures of the database `DB`. • `db:optimize("DB", true(), map { 'ftindex': true() })` optimizes all database structures of the database `DB` and creates a full-text index.

db:rename

Signatures	`db:rename($db as xs:string, $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	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0008`: new document names would 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 xs:string, $path as xs:string, $input as item()) as empty-sequence()` `db:replace($db as xs:string, $path as xs:string, $input as item(), $options as item()) as empty-sequence()`
Summary	Replaces a document, specified by `$path`, in the database `$db` with the content of `$input`, or adds it as a new document.The `$options` argument can be used to control parsing. The syntax is identical to the db:create function: Allowed options are all parsing and XML parsing options.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0014`: `$path` points to a directory.`FODC0002`: `$input` is a string representing a path, which cannot be read.`FOUP0001`: `$input` is neither a string nor 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.
	
	The following query can be used to import files from a directory to a database:
	
	`let $source := '/home/john/xml/source'`
	`for $file in file:list($source, true())`
	`let $path := $source || $file`
	`where not(file:is-dir($path))`
	`return db:replace('db', $file, doc($path))`

db:store

Signatures	`db:store($db as xs:string, $path as xs:string, $input as item()) as empty-sequence()`
Summary	Stores a binary resource specified by `$input` in the database `$db` and the location specified by `$path`.

Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0003`: the database is not *persistent* (stored on disk).`FODC0007`: the specified path is invalid.`FOUP0002`: 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

Signatures	`db:output($result 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 items will be cached and returned after the updates on the *pending update list* have been processed. As nodes may be updated, they will be copied before being cached.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.

db:output-cache

Introduced with Version 8.2:

Signatures	`db:output-cache() as item()*`
Summary	Returns the items that have been cached by db:output. It can be used to check which items will eventually be returned as result of an updating function.This function is non-deterministic: Its will return different results before and after items have been cached. It is e. g. useful when writing unit tests.

db:flush

Signatures	`db:flush($db as xs:string) as empty-sequence()`
Summary	Explicitly flushes the buffers of the database `$db`. This command is only useful if AUTOFLUSH has been set to `false`.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.

Helper Functions

db:name

Signatures	`db:name($node as node()) as xs:string`
Summary	Returns the name of the database in which the specified database node `$node` is stored.
Errors	`BXDB0001`: `$nodes` contains a node which is not stored in a database.

db:path

Signatures	`db:path($node as node()) as xs:string`
Summary	Returns the path of the database document in which the specified database node `$node` is stored.
Errors	`BXDB0001`: `$nodes` contains a node which is not stored in a database.

db:exists

Signatures	`db:exists($db as xs:string) as xs:boolean db:exists($db as xs:string, $path as xs:string) as xs:boolean`

Summary	Checks if the database `$db` or the resource specified by `$path` exists. `false` is returned if a database directory has been addressed.
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 xs:string, $path as xs:string) as xs:boolean`
Summary	Checks if the specified resource in the database `$db` and the path `$path` exists, and if it is a binary resource.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Examples	• `db:is-raw("DB", "music/01.mp3")` returns `true`.

db:is-xml

Signatures	`db:is-xml($db as xs:string, $path as xs:string) as xs:boolean`
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
Summary	Checks if the specified resource in the database `$db` and the path `$path` 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 xs:string, $path as xs:string) as xs:string`
Summary	Retrieves the content type of a resource in the database `$db` and the path `$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	`BXDB0002`: The addressed database does not exist or could not be opened.`FODC0002`: 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`.

Errors

Code	Description
`BXDB0001`	The referenced XML node is no database node, i.e. it is neither stored in a database nor represented as database fragment.
`BXDB0002`	The addressed database does not exist or could not be opened.
`BXDB0003`	The addressed database is not *persistent* (stored on disk).
`BXDB0004`	The database lacks an index structure required by the called function.
`BXDB0005`	A query is expected to exclusively return database nodes of a single database.
`BXDB0006`	A database path addressed with `doc()` contains more than one document.
`BXDB0007`	A database cannot be updated because it is opened by another process.

BXDB0008	Database paths cannot be renamed to empty strings.
BXDB0009	The addressed database id or pre value is out of range.
BXDB0011	The name of the specified database is invalid.
BXDB0012	A database can only be created once.
BXDB0013	The number of specified inputs and paths differs.
BXDB0014	Path points to a directory.

Changelog

Version 8.2

- Added: db:output-cache
- Removed: db:event

Version 7.9

- Updated: parsing options added to db:create, db:add and db:replace.
- Updated: allow UPDINDEX if `$all` is `true`.

Version 7.8.2

- Added: db:alter, db:copy, db:create-backup, db:drop-backup, db:restore

Version 7.8

- Removed: db:fulltext (use ft:search instead)

Version 7.7

- Added: db:export, db:name, db:path
- Updated: `$options` argument added to db:create and db:optimize.
- Updated: the functions no longer accept Database Nodes as reference. Instead, the name of a database must now be specified.

Version 7.6

- Updated: db:create: allow more than one input and path.

Version 7.5

- Updated: db:add: input nodes will be automatically converted to document nodes
- Added: db:backups
- Added: db:create
- Added: db:drop

Version 7.3

- Added: db:flush

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

Chapter 40. Fetch Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides simple functions to fetch the content of resources identified by URIs. Resources can be stored locally or remotely and e.g. use the `file://` or `http://` scheme. If more control over HTTP requests is required, the HTTP Module can be used. With the HTML Module, retrieved HTML documents can be converted to XML.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/fetch` namespace, which is statically bound to the `fetch` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

fetch:binary

Signatures	`fetch:binary($uri as xs:string) as xs:base64Binary`
Summary	Fetches the resource referred to by the given URI and returns it as streamable `xs:base64Binary`.
Errors	`BXFE0001`: the URI could not be resolved, or the resource could not be retrieved.
Examples	• `fetch:binary("http://images.trulia.com/blogimg/c/5/f/4/679932_1298401950553_o.jpg")` returns the addressed image. • `stream:materialize(fetch:binary("http://en.wikipedia.org"))` returns a materialized representation of the streamable result.

fetch:text

Signatures	`fetch:text($uri as xs:string) as xs:string` `fetch:text($uri as xs:string, $encoding as xs:string) as xs:string`
Summary	Fetches the resource referred to by the given URI and returns it as streamable `xs:string`.
Errors	`BXFE0001`: the URI could not be resolved, or the resource could not be retrieved. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.`BXFE0002`: the specified encoding is not supported, or unknown.
Examples	• `fetch:text("http://en.wikipedia.org")` returns a string representation of the English Wikipedia main HTML page. • `stream:materialize(fetch:text("http://en.wikipedia.org"))` returns a materialized representation of the streamable result.

fetch:xml

Signatures	`fetch:xml($uri as xs:string) as document-node()` `fetch:xml($uri as xs:string, $options as item()) as document-node()`
Summary	Fetches the resource referred to by the given `$uri` and returns it as an XML document.In contrast to `fn:doc`, each function call returns a different document node. As a consequence, document instances created by this function will not be kept in memory until the end of query evaluation.The `$options` argument can be used to change the parsing behavior. Allowed options are all parsing and XML parsing options in lower case. Options can be specified either... • as children of an `<options/>` element, e.g.

```
<options>
  <chop value='false'/>
</options>
```

- or as map, which contains all key/value pairs:

```
map { "chop": false() }
```

Errors	`BXFE0001`: the URI could not be resolved, or the resource could not be retrieved.
Examples	• `fetch:xml("http://en.wikipedia.org")` returns an XML representation of the English Wikipedia main HTML page.

fetch:content-type

Signatures	`fetch:content-type($uri as xs:string) as xs:string`
Summary	Returns the content-type (also called mime-type) of the resource specified by `$uri`: • If a remote resource is addressed, the request header will be evaluated. • If the addressed resource is locally stored, the content-type will be guessed based on the file extension.
Errors	`BXFE0001`: the URI could not be resolved, or the resource could not be retrieved.
Examples	• `fetch:content-type("http://docs.basex.org/skins/vector/images/wiki.png")` returns `image/png`.

Errors

Code	Description
`BXFE0001`	The URI could not be resolved, or the resource could not be retrieved.
`BXFE0002`	The specified encoding is not supported, or unknown.

Changelog

Version 8.0

- Added: fetch:xml

The module was introduced with Version 7.6.

Chapter 41. File Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions related to file system operations, such as listing, reading, or writing files.

This module is based on the EXPath File Module.

Conventions

All functions and errors in this module are assigned to the `http://expath.org/ns/file` namespace, which is statically bound to the `file` prefix.

For serialization parameters, the `http://www.w3.org/2010/xslt-xquery-serialization` namespace is used, which is statically bound to the `output` prefix.

Returned strings that refer to existing directories are suffixed with a directory separator. The error `invalid-path` is raised if a path is invalid.

Read Operations

file:list

Signatures	`file:list($dir as xs:string) as xs:string*` `file:list($dir as xs:string, $recursive as xs:boolean) as xs:string*` `file:list($dir as xs:string, $recursive as xs:boolean, $pattern as xs:string) as xs:string*`
Summary	Lists all files and directories found in the specified `$dir`. 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. If present, only those files and directories are returned that correspond to the pattern. Several patterns can be separated with a comma (`,`).
Errors	`not-found`: the specified file does not exist.`no-dir`: the specified path does not point to a directory.`io-error`: the operation fails for some other reason.

file:children

Signatures	`file:children($dir as xs:string) as xs:string*`
Summary	Returns the full paths to all files and directories found in the specified `$dir`.The inverse function is file:parent. The related function file:list returns relative file paths.
Errors	`not-found`: the specified file does not exist.`no-dir`: the specified path does not point to a directory.`io-error`: the operation fails for some other reason.

file:read-binary

Signatures	`file:read-binary($path as xs:string) as xs:base64Binary` `file:read-binary($path as xs:string, $offset as xs:integer) as xs:base64Binary` `file:read-binary($path as xs:string, $offset as xs:integer, $length as xs:integer) as xs:base64Binary`
Summary	Reads the binary content of the file specified by `$path` and returns it as streamable `xs:base64Binary`.The optional parameters `$offset` and `$length` can be used to read chunks of a file.
Errors	`not-found`: the specified file does not exist.`is-dir`: the specified path is a directory.`out-of-range`: the offset or length is negative, or the chosen values would exceed the file bounds.`io-error`: the operation fails for some other reason.

Examples	• `stream:materialize(file:read-binary("config.data"))` returns a materialized representation of the streamable result.

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 streamable `xs:string`.The optional parameter `$encoding` defines the encoding of the file.
Errors	`not-found`: the specified file does not exist.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: the operation fails for some other reason. Invalid XML characters will be ignored if the `CHECKSTRINGS` option is turned off.
Examples	• `stream:materialize(file:read-text("config.txt"))` returns a materialized representation of the streamable result.

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	`not-found`: the specified file does not exist.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: the operation fails for some other reason.

Write Operations

file:create-dir

Signatures	`file:create-dir($dir as xs:string) as empty-sequence()`
Summary	Creates the directory specified by `$dir`, including all non-existing parent directories.
Errors	`exists`: a file with the same path already exists.`io-error`: the operation fails for some other reason.

file:create-temp-dir

Signatures	`file:create-temp-dir($prefix as xs:string, $suffix as xs:string) as xs:string` `file:create-temp-dir($prefix as xs:string, $suffix as xs:string, $dir as xs:string) as xs:string`
Summary	Creates a new temporary directory that did not exist before this function was called, and returns its full file path. The directory name begins and ends with the specified `$prefix` and `$suffix`. If no directory is specified via `$dir`, the directory will be placed in the system's default temporary directory. The operation will create all non-existing parent directories.
Errors	`no-dir`: the specified directory points to a file.`io-error`: the directory could not be created.

file:create-temp-file

Signatures	`file:create-temp-file($prefix as xs:string, $suffix as xs:string) as xs:string` `file:create-temp-file($prefix as xs:string, $suffix as xs:string, $dir as xs:string) as xs:string`
Summary	Creates a new temporary file that did not exist before this function was called, and returns its full file path. The file name begins and ends with the specified `$prefix` and `$suffix`. If no directory is

	specified via `$dir`, the file will be placed in the system's default temporary directory. The operation will create all non-existing parent directories.
Errors	`no-dir`: the specified directory points to a file.`io-error`: the directory could not be created.

file:delete

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	`not-found`: the specified path does not exist.`io-error`: 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 item()) as empty-sequence()`

Summary	Writes a serialized sequence of items to the specified file. If the file already exists, it will be overwritten.The `$params` argument contains serialization parameters (see Serialization for more details), which can either be specified
	• as children of an `<output:serialization-parameters/>` element, as defined for the fn:serialize() function; e.g.:
	`<output:serialization-parameters>`
	`  <output:method value='xml'/>`
	`  <output:cdata-section-elements value="div"/>`
	`  ...`
	`</output:serialization-parameters>`
	• as map, which contains all key/value pairs:
	`map { "method": "xml", "cdata-section-elements": "div", ... }`

Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`io-error`: the operation fails for some other reason.

file:write-binary

Signatures	`file:write-binary($path as xs:string, $value as xs:anyAtomicType) as empty-sequence()` `file:write-binary($path as xs:string, $value as xs:anyAtomicType, $offset as xs:integer) as empty-sequence()`
Summary	Writes a binary item (xs:base64Binary, xs:hexBinary) to the specified file. If the file already exists, it will be overwritten.If `$offset` is specified, data will be written at this file position. An existing file may be resized by that operation.
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`out-of-range`: the offset is negative, or it exceeds the current file size.`io-error`: the operation fails for some other reason.

file:write-text

Signatures	`file:write-text($path as xs:string, $value as xs:string) as empty-sequence()` `file:write-text($path as xs:string, $value as xs:string, $encoding as xs:string) as empty-sequence()`

Summary	Writes a string to the specified file. If the file already exists, it will be overwritten.The optional parameter `$encoding` defines the output encoding (default: UTF-8).
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: the operation fails for some other reason.

file:write-text-lines

Signatures	`file:write-text-lines($path as xs:string, $values as xs:string*) as empty-sequence()` `file:write-text-lines($path as xs:string, $values as xs:string*, $encoding as xs:string) as empty-sequence()`
Summary	Writes a sequence of strings to the specified file, each followed by the system specific newline character. If the file already exists, it will be overwritten.The optional parameter `$encoding` defines the output encoding (default: UTF-8).
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: 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 item()) as empty-sequence()`
Summary	Appends a serialized sequence of items to the specified file. If the file does not exists, a new file is created.
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`io-error`: the operation fails for some other reason.

file:append-binary

Signatures	`file:append-binary($path as xs:string, $value as xs:anyAtomicType) as empty-sequence()`
Summary	Appends a binary item (xs:base64Binary, xs:hexBinary) to the specified file. If the file does not exists, a new one is created.
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`io-error`: the operation fails for some other reason.

file:append-text

Signatures	`file:append-text($path as xs:string, $value as xs:string) as empty-sequence()` `file:append-text($path as xs:string, $value as xs:string, $encoding as xs:string) as empty-sequence()`
Summary	Appends a string to a file specified by `$path`. If the specified file does not exists, a new file is created.The optional parameter `$encoding` defines the output encoding (default: UTF-8).
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: the operation fails for some other reason.

file:append-text-lines

Signatures	`file:append-text-lines($path as xs:string, $values as xs:string*) as empty-sequence()` `file:append-text-lines($path as xs:string, $values as xs:string*, $encoding as xs:string) as empty-sequence()`

Summary	Appends a sequence of strings to the specified file, each followed by the system specific newline character. If the specified file does not exists, a new file is created.The optional parameter `$encoding` defines the output encoding (default: UTF-8).
Errors	`no-dir`: the parent of specified path is no directory.`is-dir`: the specified path is a directory.`unknown-encoding`: the specified encoding is not supported, or unknown.`io-error`: the operation fails for some other reason.

file:copy

Signatures	`file:copy($source as xs:string, $target as xs:string) as empty-sequence()`
Summary	Copies a file or directory specified by `$source` to the file or directory specified by `$target`. If the target file already exists, it will be overwritten. No operation will be performed if the source and target path are equal.
Errors	`not-found`: the specified source does not exist.`exists`: the specified source is a directory and the target is a file.`no-dir`: the parent of the specified target is no directory.`io-error`: 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`. If the target file already exists, it will be overwritten. No operation will be performed if the source and target path are equal.
Errors	`not-found`: the specified source does not exist.`exists`: the specified source is a directory and the target is a file.`no-dir`: the parent of the specified target is no directory.`io-error`: the operation fails for some other reason.

File Properties

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-dir

Signatures	`file:is-dir($path as xs:string) as xs:boolean`
Summary	Returns an `xs:boolean` indicating whether the argument `$path` points to an existing directory.

file:is-absolute

Signatures	`file:is-absolute($path as xs:string) as xs:boolean`
Summary	Returns an `xs:boolean` indicating whether the argument `$path` is absolute.The behavior of this function depends on the operating system: On Windows, an absolute path starts with the drive letter and a colon, whereas on Linux it starts with a slash.

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	`not-found`: 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`, or `0` for directories.
Errors	`not-found`: the specified file does not exist.

Path Functions

file:name

Signatures	`file:name($path as xs:string) as xs:string`
Summary	Returns the name of a file or directory specified by `$path`. An empty string is returned if the path points to the root directory.

file:parent

Signatures	`file:parent($path as xs:string) as xs:string?`
Summary	Returns the absolute path to the parent directory of a file or directory specified by `$path`. An empty sequence is returned if the path points to a root directory.The inverse function is file:children.
Examples	• `file:parent(static-base-uri())` returns the directory of the current XQuery module.

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	`not-found`: the specified file does not exist.`io-error`: the specified path cannot be transformed to its native representation.

file:resolve-path

Updated in BaseX 8.2: base argument added

Signatures	`file:resolve-path($path as xs:string) as xs:string` `file:resolve-path($path as xs:string, $base as xs:string) as xs:string`
Summary	Transforms the `$path` argument to an absolute operating system path.If the path is relative, and if an absolute `$base` path is specified, it will be resolved against this path.
Errors	`is-relative`: the specified base path is relative.

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.

System Properties

file:dir-separator

Signatures	`file:dir-separator() as xs:string`
Summary	Returns the directory separator used by the operating system, such as `/` or `\`.

file:path-separator

Signatures	`file:path-separator() as xs:string`
Summary	Returns the path separator used by the operating system, such as `;` or `:`.

file:line-separator

Signatures	`file:line-separator() as xs:string`
Summary	Returns the line separator used by the operating system, such as `
`, `
` or `
`.

file:temp-dir

Signatures	`file:temp-dir() as xs:string`
Summary	Returns the system's default temporary-file directory.

file:current-dir

Signatures	`file:current-dir() as xs:string`
Summary	Returns the current working directory. - This function returns the same result as the function call `file:resolve-path()`<nulli/> </td>

file:base-dir

Signatures	`file:base-dir() as xs:string?`
Summary	Returns the parent directory of the static base URI. If the Base URI property is undefined, the empty sequence is returned. - If a static base URI exists, and if points to a local file path, this function returns the same result as the expression `file:parent(static-base-uri())`.

Errors

Code	Description
`not-found`	A specified path does not exist.
`invalid-path`	A specified path is invalid.
`exists`	A file with the same path already exists.
`no-dir`	The specified path does not point to a directory.
`is-dir`	The specified path is a directory.
`is-relative`	The specified path is relative (and must be absolute).
`unknown-encoding`	The specified encoding is not supported, or unknown.
`out-of-range`	The specified offset or length is negative, or the chosen values would exceed the file bounds.
`io-error`	The operation fails for some other reason specific to the operating system.

Changelog

Version 8.2

- Added: file:is-absolute
- Updated: file:resolve-path: base argument added

Version 8.0

- Added: file:current-dir, file:base-dir, file:children

Version 7.8

- Added: file:parent, file:name
- Updated: error codes; file:read-binary, file:write-binary: `$offset` and `$length` arguments added.
- Deleted: file:base-name, file:dir-name

Version 7.7

- Added: file:create-temp-dir, file:create-temp-file, file:temp-dir
- Updated: all returned strings that refer to existing directories will be suffixed with a directory separator.

Version 7.3

- Added: file:append-text, file:write-text, file:append-text-lines, file:write-text-lines, file:line-separator
- Aligned with latest specification: $file:directory-separator → file:dir-separator, $file:path-separator → file:path-separator, file:is-directory → file:is-dir, file:create-directory → file:create-dir
- Updated: file:write-binary, file:append-binary: output limited to a single value

Version 7.2.1

- Updated: file:delete: `$recursive` parameter added to prevent sub-directories from being accidentally deleted.
- Fixed: file:list now returns relative instead of absolute paths.

Chapter 42. Full-Text Module

Read this entry online in the BaseX Wiki.

This XQuery Module extends the W3C Full Text Recommendation 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.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/ft` namespace, which is statically bound to the `ft` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

ft:search

Signatures	`ft:search($db as xs:string, $terms as item()*) as text()*` `ft:search($db as xs:string, $terms as item()*, $options as item()) as text()*`

Summary	Returns all text nodes from the full-text index of the database `$db` that contain the specified `$terms`.The options used for tokenizing the input and building the full-text will also be applied to the search terms. As an example, if the index terms have been stemmed, the search string will be stemmed as well. The `$options` argument can be used to control full-text processing. Options can be either specified
	
	• as children of an `<options/>` element, e.g.:
	
	`<options>`
	`  <key1 value='value1'/>`
	`  ...`
	`</options>`
	
	• as map, which contains all key/value pairs:
	
	`map { "key1": "value1", ... }`
	
	The following options are supported (the introduction on Full-Text processing gives you equivalent expressions in the XQuery Full-Text notation):
	
	• `mode` : determines the mode how tokens are searched. Allowed values are `any`, `any word`, `all`, `all words`, and `phrase`. `any` is the default search mode.
	
	• `fuzzy` : turns fuzzy querying on or off. Allowed values are `true` and `false`. By default, fuzzy querying is turned off.
	
	• `wildcards` : turns wildcard querying on or off. Allowed values are `true` and `false`. By default, wildcard querying is turned off.
	
	• `ordered` : requires that all tokens occur in the order in which they are specified. Allowed values are `true` and `false`. The default is `false`.
	
	• `content` : specifies that the matched tokens need to occur at the beginning or end of a searched string, or need to cover the entire string. Allowed values are `start`, `end`, and `entire`. By default, the option is turned off.

- `scope` : defines the scope in which tokens must be located. The option has following sub options:
  - `same` : can be set to `true` or `false`. It specifies if tokens need to occur in the same or different units.
  - `unit` : can be `sentence` or `paragraph`. It specifies the unit for finding tokens.
- `window` : sets up a window in which all tokens must be located. By default, the option is turned off. It has following sub options:
  - `size` : specifies the size of the window in terms of *units*.
  - `unit` : can be `sentences`, `sentences` or `paragraphs`. The default is `words`.
- `distance` : specifies the distance in which tokens must occur. By default, the option is turned off. It has following sub options:
  - `min` : specifies the minimum distance in terms of *units*. The default is `0`.
  - `max` : specifies the maximum distance in terms of *units*. The default is `#`.
  - `unit` : can be `words`, `sentences` or `paragraphs`. The default is `words`.

Errors

`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0004`: the full-text index is not available.`BXFT0001`: the fuzzy and wildcard option cannot be both specified.

Examples

- `ft:search("DB", "QUERY")` : Return all text nodes of the database `DB` that contain the term `QUERY`.
- Return all text nodes of the database `DB` that contain the numbers `2010` and `2020`:`ft:search("DB", ("2010", "2020"), map { 'mode': 'all' })`
- Return text nodes that contain the terms `A` and `B` in a distance of at most 5 words:

```
ft:search("db", ("A", "B"), map {
  "mode": "all words",
  "distance": {
    "max": "5",
    "unit": "words"
  }
})
```

- Iterate over three databases and return all elements containing terms similar to `Hello World` in the text nodes:

```
let $terms := "Hello Worlds"
let $fuzzy := true()
let $options := <options><fuzzy value="{ $fuzzy }"/></options>
for $db in 1 to 3
let $dbname := 'DB' || $db
return ft:search($dbname, $terms, $options)/..
```

ft:contains

Signatures

`ft:contains($input as item()*, $terms as item()*) as xs:boolean`
`ft:contains($input as item()*, $terms as item()*, $options as item()) as xs:boolean`

Summary

Checks if the specified `$input` items contain the specified `$terms`.The function does the same as the Full-Text expression `contains text`, but options can be specified more dynamically. The `$options` are the same as for ft:search, and the following ones in addition:

- `case` : determines how character case is processed. Allowed values are `insensitive`, `sensitive`, `upper` and `lower`. By default, search is case insensitive.
- `diacritics` : determines how diacritical characters are processed. Allowed values are `insensitive` and `sensitive`. By default, search is diacritical insensitive.
- `stemming` : determines is tokens are stemmed. Allowed values are `true` and `false`. By default, stemming is turned off.
- `language` : determines the language. This option is relevant for stemming tokens. All language codes are supported. The default language is `en`.

Errors

`BXFT0001`: the fuzzy and wildcard option cannot be both specified.

Examples

- Checks if `jack` or `john` occurs in the input string `John Doe`:

```
ft:contains("John Doe", ("jack", "john"), map { "mode": "any" })
```

- Calls the function with stemming turned on and off:

```
(true(), false()) ! ft:contains("Häuser", "Haus", map { 'stemming': .,
 'language':'de' })
```

ft:mark

Signatures

`ft:mark($nodes as node()*) as node()*` `ft:mark($nodes as node()*, $name as xs:string) as node()*`

Summary

Puts a marker element around the resulting `$nodes` of a full-text index request.The default name of the marker element is `mark`. An alternative name can be chosen via the optional `$name` argument.Please note that:

- the full-text expression that computes the token positions must be specified as argument of the `ft:mark()` function, as all position information is lost in subsequent processing steps. You may need to specify more than one full-text expression if you want to use the function in a FLWOR expression, as shown in Example 2.
- 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, as shown in Example 3.

Examples

Example 1: 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'])
```

Example 2: The following expression loops through the first ten full-text results and marks the results in a second expression:

```
let $start := 1
let $end   := 10
let $term  := 'welcome'
for $ft in (db:open('DB')//*[text() contains text { $term }])[position()
 = $start to $end]
return element hit {
  ft:mark($ft[text() contains text { $term }])
}
```

Example 3: The following expression returns `<p><b>word</b></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()*, $name as xs:string) as node()*` `ft:extract($nodes as node()*, $name 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 `$name` argument.The default length of the returned text is `150` characters. An alternative length can be specified via the optional `$length` argument. Note that the effective text length may differ from the specified text due to formatting and readibility issues.For more details on this function, please have a look at ft:mark.
Examples	• The following query may return `<XML>...<b>hello</b>...<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.
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($db as xs:string) as element(value)*` `ft:tokens($db as xs:string, $prefix as xs:string) as element(value)*`
Summary	Returns all full-text tokens stored in the index of the database `$db`, along with their numbers of occurrences.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	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0004`: the full-text index is not available.
Examples	Finds the number of occurrences for a single, specific index entry (the positional predicate speeds up retrieval):

```
let $term := ft:tokenize($term)
return data((ft:tokens('db', $term)[. = $term])[1]/@count)
```

ft:tokenize

Signatures	`ft:tokenize($input as xs:string) as xs:string*` `ft:tokenize($input as xs:string, $options as item()) as xs:string*`

Summary	Tokenizes the given `$input` string, using the current default full-text options or the `$options` specified as second argument. The following options are available:
	• `case` : determines how character case is processed. Allowed values are `insensitive`, `sensitive`, `upper` and `lower`. By default, search is case insensitive.
	• `diacritics` : determines how diacritical characters are processed. Allowed values are `insensitive` and `sensitive`. By default, search is diacritical insensitive.
	• `stemming` : determines is tokens are stemmed. Allowed values are `true` and `false`. By default, stemming is turned off.
	• `language` : determines the language. This option is relevant for stemming tokens. All language codes are supported. The default language is `en`.
	The `$options` argument can be used to control full-text processing. Options can be either specified
	• as children of an `<options/>` element, e.g.:
	`<options><key1 value='value1'/>...</options>`
	• as map, which contains all key/value pairs:
	`map { "key1": "value1", ... }`

Examples	• `ft:tokenize("No Doubt")` returns the two strings `no` and `doubt`. • `ft:tokenize("École", map { 'diacritics': 'sensitive' })` returns the string `école`. • `declare ft-option using stemming; ft:tokenize("GIFTS")` returns a single string `gift`.

ft:normalize

Signatures	`ft:normalize($input as xs:string) as xs:string*` `ft:normalize($input as xs:string, $options as item()) as xs:string*`
Summary	Normalizes the given `$input` string, using the current default full-text options or the `$options` specified as second argument. The function provides the same arguments as ft:tokenize.
Examples	• `ft:tokenize("Häuser am Meer", map { 'case': 'sensitive' })` returns the string `Hauser am Meer`.

Errors

Code	Description
`BXFT0001`	Both wildcards and fuzzy search have been specified as search options.

Changelog

Version 8.0

- Added: ft:contains, ft:normalize

- Updated: Options added to ft:tokenize

Version 7.8

- Added: ft:contains
- Updated: Options added to ft:search

Version 7.7

- Updated: the functions no longer accept Database Nodes as reference. Instead, the name of a database must now be specified.

Version 7.2

- Updated: ft:search (second argument generalized, third parameter added)

Version 7.1

- Added: ft:tokens, ft:tokenize

Chapter 43. Geo Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions that may be applied to geometry data conforming to the Open Geospatial Consortium (OGC) Simple Feature (SF) data model. It is based on the EXPath Geo Module and uses the JTS library.

Geometries introduced in GML 2 are: Point, LineString, LinearRing, Polygon, MultiPoint, MultiLineString, MultiPolygon, and MultiGeometry. All nodes queried by BaseX should be a valid geometry. The only geometry type which is not supported by BaseX right now is MultiGeometry.

Conventions

- This module is included in the complete distributions of BaseX (zip, exe, war).
- All functions are assigned to the `http://expath.org/ns/geo` namespace, which must be dynamically imported:

```
import module namespace geo = "http://expath.org/ns/geo";
...
```

- In this documentation, the namespace is bound to the `geo` prefix.
- All errors are assigned to the `http://expath.org/ns/error` namespace, which is statically bound to the `experr` prefix.

General Functions

geo:dimension

Signatures	`geo:dimension($geometry as element(*)) as xs:integer`
Summary	Returns the dimension of the given geometry `$geometry`.
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.

Example	`import module namespace geo = "http://expath.org/ns/geo";`
	`declare namespace gml='http://www.opengis.net/gml';`
	
	`let $point := <gml:Point><gml:coordinates>1,2</gml:coordinates></gml:Point>`
	`return geo:dimension($point)`

geo:geometry-type

Signatures	`geo:geometry-type($geometry as element(*)) as xs:QName`
Summary	Returns the name of the geometry type of given geometry `$geometry`, if the geometry is not recognized with an error massage.
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.

Example	**Query:**
	
	`import module namespace geo = "http://expath.org/ns/geo";`
	`declare namespace gml='http://www.opengis.net/gml';`

```
let $point := <gml:Point><gml:coordinates>1,2</gml:coordinates></
gml:Point>
return geo:geometry-type($point)
```

Result:

```
gml:Point
```

geo:srid

Signatures	`geo:srid($geometry as element(*)) as xs:integer`
Summary	Returns the ID of the Spatial Reference System used by the given geometry `$geometry`. Spatial Reference System information is supported in the simple way defined in the SFS. A Spatial Reference System ID (SRID) is present in each Geometry object. Geometry provides basic accessor operations for this field, but no others. The SRID is represented as an integer (based on the OpenGIS Simple Features Specifications For SQL). Here is a difference between the EXPath Geo Module and the implementation in BaseX, since the specification return the URI.
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.

geo:envelope

Signatures	`geo:envelope($geometry as element(*)) as element(*)`
Summary	Returns the gml:Envelope of the given geometry `$geometry`. The envelope is the minimum bounding box of this geometry. If this Geometry is the empty geometry, returns an empty Point. If the Geometry is a point, returns a non-empty Point. Otherwise, returns a Polygon whose points are (minx, miny), (maxx, miny), (maxx, maxy), (minx, maxy), (minx, miny).
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $line := <gml:LinearRing><gml:coordinates>1,1 20,1 20,20 1,20 1,1</
gml:coordinates></gml:LinearRing>
return geo:envelope($line)
```

geo:as-text

Signatures	`geo:as-text($geometry as element(*)) as xs:string`
Summary	Returns the WKT (Well-known Text) representation of the given geometry `$geometry`. The envelope is the minimum bounding box of this geometry
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $point := <gml:Point><gml:coordinates>1,2</gml:coordinates></
gml:Point>
```

```
return geo:as-text($point)
```

Result:

```
POINT (1 2)
```

geo:as-binary

Signatures	`geo:as-binary($geometry as element(*)) as xs:base64Binary`
Summary	Returns the WKB (Well-known Binary) representation of the given geometry `$geometry`.
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $point := <gml:Point><gml:coordinates>1,2</gml:coordinates></
gml:Point>
return geo:as-text($point)
```

Result:

```
AAAAAAE/8AAAAAAAAEAAAAAAAAAA
```

geo:is-simple

Signatures	`geo:is-simple($geometry as element(*)) as xs:boolean`
Summary	Returns whether the given geometry is simple `$geometry` and does not have has no anomalous geometric points (ie. the geometry does not self-intersect and does not pass through the same point more than once (may be a ring)).
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $line := <gml:MultiLineString>
                <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>
                <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>
              </gml:MultiLineString>

return geo:is-simple($line)
```

Result:

```
true
```

geo:boundary

Signatures	`geo:boundary($geometry as element(*)) as element(*)?`

Summary	Returns the boundary of the given geometry `$geometry`, in GML 2. The return value is a sequence of either gml:Point or gml:LinearRing elements as a GeometryCollection object. For a Point or MultiPoint, the boundary is the empty geometry, nothing is returned.
Errors	`GEO0001`: the given element is not recognized as a valid geometry.`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>

return geo:boundary($Line)
```

Result:

```
<gml:MultiPoint>
  <gml:pointMember>
    <gml:Point>
      <gml:coordinates>1.0,1.0</gml:coordinates>
    </gml:Point>
  </gml:pointMember>
  <gml:pointMember>
    <gml:Point>
      <gml:coordinates>2.0,1.0</gml:coordinates>
    </gml:Point>
  </gml:pointMember>
</gml:MultiPoint>
```

geo:num-geometries

Signatures	`geo:num-geometries($geometry as element(*)) as xs:integer`
Summary	Returns the number of geometry in a geometry-collection `$geometry`, in GML. For the geometries which are not a collection, it returns the instant value 1. This function is implemented wider than the specification and accepts all types of geometries, while the specification limits it to the collection types (MultiPoint, MultiPolygon, ...).
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line :=  <gml:MultiLineString>
                <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>
                <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>
              </gml:MultiLineString>

return geo:num-geometries($Line)
```

Result:

```
2
```

Query:

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>

return geo:num-geometries($Line)
```

Result:

```
1
```

geo:geometry-n

Signatures	`geo:geometry-n($geometry as element(*), $geoNumber as xs:integer) as element(*)`
Summary	Returns the Nth geometry in geometry-collection `$geometry`, in GML. For the geometries which are not a collection, it returns the geometry if geoNumber `$geoNumber` is 1. This function is implemented wider than the specification and accepts all types of geometries, while the specification limits it to the collection types (MultiPoint, MultiPolygon, ...).
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0004`: the the input index of geometry is out of range.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line :=  <gml:MultiLineString>
                <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>
                <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>
              </gml:MultiLineString>

return geo:geometry-n($Line, 1)
```

Result:

```
<gml:LineString>
  <gml:coordinates>1.0,1.0 0.0,0.0 2.0,1.0</gml:coordinates>
</gml:LineString>
```

Query:

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>

return geo:geometry-n($Line, 1)
```

Result:

```
<gml:LineString>
  <gml:coordinates>1.0,1.0 0.0,0.0 2.0,1.0</gml:coordinates>
</gml:LineString>
```

geo:length

Signatures	`geo:length($geometry as element(*)) as xs:double`
Summary	Returns the length of the geometry `$geometry`. If the geometry is a point, zero value will be returned.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon := <gml:Polygon>
                <gml:outerBoundaryIs>
 <gml:LinearRing><gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
             </gml:LinearRing> </gml:outerBoundaryIs></gml:Polygon>

return geo:length($Polygon)
```

Result:

```
4
```

Query:

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:length($Line)
```

Result:

```
3.6502815398728847
```

geo:num-points

Signatures	`geo:num-points($geometry as element(*)) as xs:integer`
Summary	Returns integer value of number of the points in the given geometry`$geometry`. It can be used not only for Lines, also any other geometry types, like MultiPolygon. For Point geometry it will return 1. This is an implementation different from the EXPath geo specification, as it limits the input geometry type only to lines.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:num-points($Line)
```

Result:

```
3
```

geo:area

Signatures	`geo:area($geometry as element(*)) as xs:double`
Summary	Returns the area of the given geometry `$geometry`. For points and line the return value will be zero.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon := <gml:Polygon>
                <gml:outerBoundaryIs>
 <gml:LinearRing><gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
             </gml:LinearRing> </gml:outerBoundaryIs></gml:Polygon>

return geo:area($Polygon)
```

Result:

```
1
```

geo:centroid

Signatures	`geo:centroid($geometry as element(*)) as element(*)`
Summary	Returns the mathematical centroid of the given geometry `$geometry`, as a gml:Point. Based on the definition, this point is not always on the surface of the geometry `$geometry`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point :=<gml:MultiPoint>
                  <gml:Point><gml:coordinates>1,1</gml:coordinates></
gml:Point>
                  <gml:Point><gml:coordinates>10,10</gml:coordinates></
gml:Point>
                  <gml:Point><gml:coordinates>2,2</gml:coordinates></
gml:Point>
```

```
                </gml:MultiPoint>

return geo:centroid($Point)
```

Result:

```
<gml:Point>
  <gml:coordinates>4.333333333333333,4.333333333333333</gml:coordinates>
</gml:Point>
```

Query:

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon := <gml:Polygon>
                   <gml:outerBoundaryIs>
 <gml:LinearRing><gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
                </gml:LinearRing> </gml:outerBoundaryIs></gml:Polygon>

return geo:centroid($Polygon)
```

Result:

```
<gml:Point>
  <gml:coordinates>1.5,1.5</gml:coordinates>
</gml:Point>
```

geo:point-on-surface

Signatures	`geo:point-on-surface($geometry as element(*)) as element(*)`
Summary	Returns an interior point on the given geometry `$geometry`, as a gml:Point. It is guaranteed to be on surface. Otherwise, the point may lie on the boundary of the geometry.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line :=<gml:LineString><gml:coordinates>1,1 55,99 2,1</
gml:coordinates></gml:LineString>

return geo:point-on-surface($Line)
```

Result:

```
<gml:Point>
  <gml:coordinates>55.0,99.0</gml:coordinates>
</gml:Point>
```

Query:

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
```

```
let $Polygon := <gml:Polygon>
                   <gml:outerBoundaryIs>
 <gml:LinearRing><gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
                </gml:LinearRing> </gml:outerBoundaryIs></gml:Polygon>

return geo:point-on-surface($Polygon)
```

Result:

```
<gml:Point>
  <gml:coordinates>1.5,1.5</gml:coordinates>
</gml:Point>
```

Spatial Predicate Functions

geo:equals

Signatures	`geo:equals($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially equal to $geometry2 `$geometry2`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line1:= <gml:LineString><gml:coordinates>1,1 55,99 2,1</
gml:coordinates></gml:LineString>

let $Line2:= <gml:LineString><gml:coordinates>1,1 1,1 55,99 2,1</
gml:coordinates></gml:LineString>


return geo:equals($Line1, $Line2)
```

Result:

```
true
```

geo:disjoint

Signatures	`geo:disjoint($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially disjoint from $geometry2 `$geometry2` (they have no point in common, they do not intersect each other, and the DE-9IM Intersection Matrix for the two geometries is FF*FF****).
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line1:= := <gml:MultiLineString>
```

```
                    <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>
                    <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>
                </gml:MultiLineString>


let $Line2:= <gml:LineString><gml:coordinates>0,0 2,1 3,3</
gml:coordinates></gml:LineString>

return geo:disjoint($Line1, $Line2)
```

Result:

```
false
```

geo:intersects

Signatures	`geo:intersects($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially intersects $geometry2 `$geometry2`. This is true if disjoint function of the two geometries returns false.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line1:= := <gml:MultiLineString>
                    <gml:LineString><gml:coordinates>1,1 0,0 2,1</
gml:coordinates></gml:LineString>
                    <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>
                </gml:MultiLineString>


let $Line2:= <gml:LineString><gml:coordinates>0,0 2,1 3,3</
gml:coordinates></gml:LineString>

return geo:intersects($Line1, $Line2)
```

Result:

```
true
```

geo:touches

Signatures	`geo:touches($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially touches $geometry2 `$geometry2`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
```

```
declare namespace gml='http://www.opengis.net/gml';

let $Line := <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>

let $Polygon:= <gml:Polygon>
                 <gml:outerBoundaryIs>
                    <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>
                </gml:outerBoundaryIs>
              </gml:Polygon>

return geo:touches($Line, $Polygon)
```

Result:

```
true
```

geo:crosses

Signatures	`geo:crosses($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially crosses $geometry2 `$geometry2`. It means, if the geometries have some but not all interior points in common. Returns true if the DE-9IM intersection matrix for the two geometries is: `T*T******` (for P/L, P/A, and L/A situations) `T*****T**` (for L/P, A/P, and A/L situations) `0********` (for L/L situations).
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>

let $Polygon:= <gml:Polygon>
                 <gml:outerBoundaryIs>
                    <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>
                </gml:outerBoundaryIs>
              </gml:Polygon>

return geo:crosses($Line, $Polygon)
```

Result:

```
false
```

geo:within

Signatures	`geo:within($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially within $geometry2 `$geometry2`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>

let $Polygon:= <gml:Polygon>
                 <gml:outerBoundaryIs>
                    <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>
                </gml:outerBoundaryIs>
              </gml:Polygon>

return geo:within($Line, $Polygon)
```

Result:

```
false
```

geo:contains

Signatures	`geo:contains($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` spatially contains $geometry2 `$geometry2`. Returns true if within function of these two geometries also returns true.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point:= <gml:Point><gml:coordinates>1,1</gml:coordinates></
gml:Point>

let $Polygon:= <gml:Polygon>
                 <gml:outerBoundaryIs>
                    <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>
                </gml:outerBoundaryIs>
              </gml:Polygon>

return geo:contains($Polygon, $Point)
```

Result:

```
false
```

geo:overlaps

Signatures	`geo:overlaps($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean`
Summary	Returns whether geometry1 `$geometry1` is spatially overlaps $geometry2 `$geometry2`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon1:= <gml:Polygon>
                   <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>1,1 20,1 20,20
 30,20 30,30 1,30 1,1</gml:coordinates></gml:LinearRing>
                   </gml:outerBoundaryIs>
                   <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>2,2 3,2 3,3 2,3
 2,2</gml:coordinates></gml:LinearRing>
                   </gml:innerBoundaryIs>
                   <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 19,10 19,19
 10,19 10,10</gml:coordinates></gml:LinearRing>
                   </gml:innerBoundaryIs>
                </gml:Polygon>

let $Polygon2:= <gml:Polygon>
                   <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</
gml:coordinates></gml:LinearRing>
                   </gml:outerBoundaryIs>
                </gml:Polygon>


return geo:overlaps($Polygon1, $Polygon2)
```

Result:

```
false
```

geo:relate

Signatures	`geo:relate($geometry1 as element(*), $geometry2 as element(*), $intersectionMatrix as xs:string) as xs:boolean`
Summary	Returns whether relationships between the boundaries, interiors and exteriors of geometry1 `$geometry1` and geometry2 `$geometry2` match the pattern specified in intersectionMatrix `$geometry2`, which should have the length of 9 charachters.The values in the DE-9IM can be T, F, *, 0, 1, 2 . - T means the intersection gives a non-empty result. - F means the intersection gives an empty result. - * means any result. - 0, 1, 2 gives the expected dimension of the result (point, curve, surface)
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point:= <gml:Point><gml:coordinates>18,11</gml:coordinates></
gml:Point>

let $Polygon:= <gml:Polygon>
                   <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 20,10 30,40
 20,40 10,10</gml:coordinates></gml:LinearRing>
                   </gml:outerBoundaryIs>
                </gml:Polygon>
```

```
return geo:relate($Point, $Polygon)
```

Result:

```
true
```

Analysis Functions

geo:distance

Signatures	`geo:distance($geometry1 as element(*), $geometry2 as element(*)) as xs:double`
Summary	Returns the shortest distance, in the units of the spatial reference system of geometry1 `$geometry1`, between the geometries, where that distance is the distance between a point on each of the geometries.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LinearRing>
                <gml:coordinates>10,400 20,200 30,100 20,100 10,400</
gml:coordinates>
             </gml:LinearRing>

let $Polygon:= <gml:Polygon>
                   <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 20,10 30,40
 20,40 10,10</gml:coordinates></gml:LinearRing>
                   </gml:outerBoundaryIs>
                </gml:Polygon>


return geo:distance($Line, $Polygon)
```

Result:

```
60
```

geo:buffer

Signatures	`geo:buffer($geometry as element(*), $distance as xs:double) as element(*)`
Summary	Returns polygonal geometry representing the buffer by distance `$distance` of geometry `$geometry` a buffer area around this geometry having the given width, in the spatial reference system of geometry. The buffer of a Geometry is the Minkowski sum or difference of the geometry with a disc of radius abs(distance). The buffer is constructed using 8 segments per quadrant to represent curves.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon:= <gml:Polygon>
                   <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 20,10 30,40
 20,40 10,10</gml:coordinates></gml:LinearRing>
                   </gml:outerBoundaryIs>
               </gml:Polygon>

return geo:buffer($Polygon)
```

geo:convex-hull

Signatures	`geo:convex-hull($geometry as element(*)) as element(*)`
Summary	Returns the convex hull geometry of the given geometry `$geometry` in GML, or the empty sequence. Actually returns the object of smallest dimension possible.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LinearRing>
                <gml:coordinates>10,400 20,200 30,100 20,100 10,400</
gml:coordinates>
             </gml:LinearRing>

return geo:convex-hull($Line)
```

Result:

```
<gml:Polygon>
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>20.0,100.0 10.0,400.0 30.0,100.0 20.0,100.0</
gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
</gml:Polygon>
```

geo:intersection

Signatures	`geo:intersection($geometry1 as element(*), $geometry2 as element(*)) as element(*)?`
Summary	Returns the intersection geometry of geometry1 `$geometry1` with geometry2 `$geometry2`, in GML or empty sequence if there is no intersection of these geometries.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
```

```
let $Line:= <gml:LinearRing>
                <gml:coordinates>10,400 20,200 30,100 20,100 10,400</
gml:coordinates>
             </gml:LinearRing>
let $Point := <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></
gml:Point>

return geo:intersection($Line, $Point)
```

Result:

```
<gml:Point>
  <gml:coordinates>1.0,1.0</gml:coordinates>
</gml:Point>
```

geo:union

Signatures	`geo:union($geometry1 as element(*), $geometry2 as element(*)) as element(*)`
Summary	Returns the union geometry of geometry1 `$geometry1` with geometry2 `$geometry2`, in GML.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LinearRing>
                <gml:coordinates>10,400 20,200 30,100 20,100 10,400</
gml:coordinates>
             </gml:LinearRing>
let $Point := <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></
gml:Point>

return geo:union($Line, $Point)
```

Result:

```
<gml:LineString>
  <gml:coordinates>1.0,1.0 55.0,99.0 2.0,1.0</gml:coordinates>
</gml:LineString>
```

geo:difference

Signatures	`geo:difference($geometry1 as element(*), $geometry2 as element(*)) as element(*)?`
Summary	Returns the difference geometry of geometry1 `$geometry1` with geometry2 `$geometry2`, in GML, or empty sequence if the difference is empty, as a set of point in geometry1 `$geometry1` and not included in geometry2 `$geometry2`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point := <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></
gml:Point>
let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:difference($Point, $Line)
```

Result:

```
<gml:Point>
  <gml:coordinates>1.0,1.0</gml:coordinates>
</gml:Point>
```

geo:sym-difference

Signatures	`geo:sym-difference($geometry1 as element(*), $geometry2 as element(*)) as element(*)?`
Summary	Returns the symmetric difference geometry of geometry1 `$geometry1` with geometry2 `$geometry2`, in GML, or empty sequence if the difference is empty, as a set of point in one of the geometries and not included in the other.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point := <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></
gml:Point>
let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:sym-difference($Point, $Line)
```

Result:

```
<gml:MultiGeometry>
  <gml:geometryMember>
    <gml:Point>
      <gml:coordinates>1.0,1.0</gml:coordinates>
    </gml:Point>
  </gml:geometryMember>
  <gml:geometryMember>
    <gml:LineString>
      <gml:coordinates>2.0,1.0 3.0,3.0 4.0,4.0</gml:coordinates>
    </gml:LineString>
  </gml:geometryMember>
</gml:MultiGeometry>
```

Functions Specific to Geometry Type

geo:x

Signatures	`geo:x($point as element(*)) as xs:double`

Summary	Returns the x coordinate of point `$point`. A point has to have an x coordinate.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point := <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></
gml:Point>

return geo:x($Point)
```

Result:

```
1
```

geo:y

Signatures	`geo:y($point as element(*)) as xs:double?`
Summary	Returns the y coordinate of point `$point`. If the point does not have the y coordinate, 0 will be returned.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point := <gml:Point><gml:coordinates>1.00,2.00</gml:coordinates></
gml:Point>

return geo:y($Point)
```

Result:

```
2
```

geo:z

Signatures	`geo:z($point as element(*)) as xs:double?`
Summary	Returns the z coordinate of point `$point`. If the point does not have the y coordinate, 0 will be returned.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Point := <gml:Point><gml:coordinates>1.00,1.00,3.00</
gml:coordinates></gml:Point>

return geo:z($Point)
```

Result:

```
3
```

geo:start-point

Signatures	`geo:start-point($line as element(*)) as element(*)`
Summary	Returns the starting point of the given line `$line`. `$line` has to be a single line, LineString or LinearRing.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a line. Other geometries are not accepted.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:start-point($Line)
```

Result:

```
<gml:Point>
  <gml:coordinates>2.0,1.0</gml:coordinates>
</gml:Point>
```

geo:end-point

Signatures	`geo:end-point($line as element(*)) as element(*)`
Summary	Returns the ending point of the given line `$line`. `$line` has to be a single line, LineString or LinearRing.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a line. Other geometries are not accepted.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:end-point($Line)
```

Result:

```
<gml:Point>
  <gml:coordinates>4.0,4.0</gml:coordinates>
</gml:Point>
```

geo:is-closed

Signatures	`geo:is-closed($line as element(*)) as xs:boolean`

Summary	Returns a boolean value that shows the line `$line` is a closed loop (start point and end point are the same) or not. `$line` has to be a line, as a geometry, LineString or LinearRing, and MultiLineString.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a line. Other geometries are not accepted.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:is-closed($Line)
```

Result:

```
false
```

geo:is-ring

Signatures	`geo:is-ring($line as element(*)) as xs:boolean`
Summary	Returns a boolean value that shows the line `$line` is a ring (closed loop and single) or not. `$line` has to be a single line, as a geometry, LineString or LinearRing.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a line. Other geometries are not accepted.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:is-ring($Line)
```

Result:

```
false
```

geo:point-n

Signatures	`geo:point-n($line as element(*)) as element(*)`
Summary	Returns the Nth point in the given line `$geometry`. `$line` has to be a single line, as a geometry, LineString or LinearRing.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a line. Other geometries are not accepted.`GEO0004`: the the input index of geometry is out of range.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
```

```
declare namespace gml='http://www.opengis.net/gml';

let $Line:= <gml:LineString><gml:coordinates>2,1 3,3 4,4</
gml:coordinates></gml:LineString>

return geo:point-n($Line,1)
```

Result:

```
<gml:Point>
  <gml:coordinates>2.0,1.0</gml:coordinates>
</gml:Point>
```

geo:exterior-ring

Signatures	`geo:exterior-ring($polygon as element(*)) as element(*)`
Summary	Returns the outer ring of the given polygon `$geometry`, as a gml:LineString.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a polygon. Other geometries are not accepted.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon:= <gml:Polygon>
                  <gml:outerBoundaryIs>
                     <gml:LinearRing><gml:coordinates>10,10 20,10 30,40
 20,40 10,10</gml:coordinates></gml:LinearRing>
                  </gml:outerBoundaryIs>
              </gml:Polygon>

return geo:exterior-ring($Polygon)
```

Result:

```
<gml:LineString>
  <gml:coordinates>10.0,10.0 20.0,10.0 30.0,40.0 20.0,40.0 10.0,10.0</
gml:coordinates>
</gml:LineString>
```

geo:num-interior-ring

Signatures	`geo:num-interior-ring($polygon as element(*)) as xs:integer`
Summary	Returns the number of interior rings in the given polygon `$geometry`.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a polygon. Other geometries are not accepted.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon:= <gml:Polygon>
                  <gml:outerBoundaryIs>
```

```
                      <gml:LinearRing><gml:coordinates>1,1 20,1 20,20
 30,20 30,30 1,30 1,1</gml:coordinates></gml:LinearRing>
                  </gml:outerBoundaryIs>
                  <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>2,2 3,2 3,3 2,3
 2,2</gml:coordinates></gml:LinearRing>
                  </gml:innerBoundaryIs>
                  <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 19,10 19,19
 10,19 10,10</gml:coordinates></gml:LinearRing>
                  </gml:innerBoundaryIs>
               </gml:Polygon>

return geo:num-interior-ring($Polygon)
```

Result:

```
2
```

geo:interior-ring-n

Signatures	`geo:interior-ring-n($polygon as element(*)) as element(*)`
Summary	Returns the outer ring of the given polygon `$geometry`, as a gml:LineString.
Errors	`GEO0001`: the given element(s) is not recognized as a valid geometry (QName).`GEO0002`: the given element cannot be read by reader for some reason.`GEO0003`: the given element has to be a polygon. Other geometries are not accepted.`GEO0004`: the the input index of geometry is out of range.`GEO0005`: the output object cannot be written as an element by writer for some reason.
Example	**Query:**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $Polygon:= <gml:Polygon>
                  <gml:outerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>1,1 20,1 20,20
 30,20 30,30 1,30 1,1</gml:coordinates></gml:LinearRing>
                  </gml:outerBoundaryIs>
                  <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>2,2 3,2 3,3 2,3
 2,2</gml:coordinates></gml:LinearRing>
                  </gml:innerBoundaryIs>
                  <gml:innerBoundaryIs>
                      <gml:LinearRing><gml:coordinates>10,10 19,10 19,19
 10,19 10,10</gml:coordinates></gml:LinearRing>
                  </gml:innerBoundaryIs>
               </gml:Polygon>

return geo:interior-ring-n($Polygon, 1)
```

Result:

```
<gml:LineString>
  <gml:coordinates>2.0,2.0 3.0,2.0 3.0,3.0 2.0,3.0 2.0,2.0</
gml:coordinates>
</gml:LineString>
```

Errors

Code	Description

GEO0001	Unrecognized Geo type.
GEO0002	The input GML node cannot be read by GMLreader.
GEO0003	Input geometry is not an appropriate geometry for this function.
GEO0004	The input index is out of range.
GEO0005	The result geometry can not be written by GMLwriter.

Changelog

The module was introduced with Version 7.6.

Chapter 44. Hashing Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions that perform different hash operations.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/hash` namespace, which is statically bound to the `hash` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

hash:md5

Signatures	`hash:md5($value as xs:anyAtomicType) as xs:base64Binary`
Summary	Computes the MD5 hash of the given `$value`, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.
Errors	`FORG0006`: the specified value is neither a string nor a binary item.
Examples	• `xs:hexBinary(hash:md5("BaseX"))` returns `0D65185C9E296311C0A2200179E479A2`. • `hash:md5(xs:base64Binary(""))` returns `1B2M2Y8AsgTpgAmY7PhCfg==`.

hash:sha1

Signatures	`hash:sha1($value as xs:anyAtomicType) as xs:base64Binary`
Summary	Computes the SHA-1 hash of the given `$value`, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.
Errors	`FORG0006`: the specified value is neither a string nor a binary item.
Examples	• `xs:hexBinary(hash:sha1("BaseX"))` returns `3AD5958F0F27D5AFFDCA2957560F121D0597A4ED`. • `hash:sha1(xs:base64Binary(""))` returns `2jmj7l5rSw0yVb/vlWAYkK/YBwk=`.

hash:sha256

Signatures	`hash:sha256($value as xs:anyAtomicType) as xs:base64Binary`
Summary	Computes the SHA-256 hash of the given `$value`, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.
Errors	`FORG0006`: the specified value is neither a string nor a binary item.
Examples	• `xs:hexBinary(hash:sha256("BaseX"))` returns `15D570763DEB75D728BB69643392873B835CCCC94A2F1E881909DA47662821A3`. • `hash:sha256(xs:base64Binary(""))` returns `47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=`.

hash:hash

Signatures	`hash:hash($value as xs:anyAtomicType, $algorithm as xs:string) as xs:base64Binary`

Summary	Computes the hash of the given `$value`, using the specified `$algorithm`. The specified values may be of type xs:string, xs:base64Binary, or xs:hexBinary.The following three algorihms are supported: `MD5`, `SHA-1`, and `SHA-256`.
Errors	`HASH0001`: the specified hashing algorithm is unknown.`FORG0006`: the specified value is neither a string nor a binary item.
Examples	• `xs:hexBinary(hash:md5("", "MD5"))` returns `D41D8CD98F00B204E9800998ECF8427E`. • `hash:md5("", "")` raises an error.

Errors

Code	Description
`HASH0001`	The specified hash algorithm is unknown.

Changelog

The module was introduced with Version 7.3.

Chapter 45. Higher-Order Functions Module

Read this entry online in the BaseX Wiki.

This XQuery Module adds some useful higher-order functions, additional to the Higher-Order Functions provided by the official specification.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/hof` namespace, which is statically bound to the `hof` prefix.

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	• `hof:id(1 to 5)` returns `1 2 3 4 5`
	• With higher-order functions:
	`let $sort-by := function($f, $seq) {`
	`  for $x in $seq`
	`  order by $f($x)`
	`  return $x`
	`  }`
	`let $sort := $sort-by(hof:id#1, ?),`
	`  $reverse-sort := $sort-by(function($x) { -$x }, ?)`
	`return (`
	`  $sort((1, 5, 3, 2, 4)),`
	`  '|',`
	`  $reverse-sort((1, 5, 3, 2, 4))`
	`)`
	returns: `1 2 3 4 5 | 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($seq as item()+, $f as function(item()*, item()) as item()*) as item()*`
Summary	Works the same as fn:fold-left, but doesn't need a seed, because the sequence must be non-empty.
Examples	• `hof:fold-left1(1 to 10, function($a, $b) { $a + $b })` returns `55`. • `hof:fold-left1((), function($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:scan-left

Signatures

`hof:scan-left($seq as item()*, $start as item()*, $f as function(item()*, item()) as item()*) as item()*`

Summary

This function is similar to fn:fold-left, but it returns a list of successive reduced values from the left. It is equivalent to:

```
declare function hof:scan-left($seq, $acc, $f) {
  if(empty($seq)) then $acc else (
    $acc,
    hof:scan-left(tail($seq), $f($acc, head($seq)), $f)
  )
};
```

Examples

- Returns triangular numbers:

```
hof:scan-left(1 to 10, 0, function($a, $b) { $a + $b })
```

hof:take-while

Signatures

`hof:take-while($seq as item()*, $pred as function(item()) as xs:boolean) as item()*`

Summary

The function returns items of `$seq` as long as the predicate `$pred` is satisfied. It is equivalent to:

```
declare function hof:take-while($seq, $pred) {
  if(empty($seq) or not($pred(head($seq)))) then () else (
    head($seq),
    hof:take-while(tail($seq), $pred)
  )
};
```

Examples

- Computes at most 100 random integers, but stops if an integer is smaller than 10:

```
hof:take-while(
  for $i in 1 to 100
  return random:integer(50),
  function($x) { $x >= 10 }
)
```

hof:top-k-by

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]
```

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

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 8.1

- Added: hof:scan-left, hof:take-while

Version 7.2

- Added: hof:top-k-by, hof:top-k-with
- Removed: hof:iterate

Version 7.0

- module added

Chapter 46. HTML Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for converting HTML to XML. Conversion will only take place if TagSoup is included in the classpath (see HTML Parsing for more details).

Conventions

All functions in this module are assigned to the `http://basex.org/modules/html` namespace, which is statically bound to the `html` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

html:parser

Signatures	`html:parser() as xs:string`
Summary	Returns the name of the applied HTML parser (currently: `TagSoup`). If an *empty string* is returned, TagSoup was not found in the classpath, and the input will be treated as well-formed XML.

html:parse

Signatures	`html:parse($input as xs:anyAtomicType) as document-node()` `html:parse($input as xs:anyAtomicType, $options as item()) as document-node()`

Summary	Converts the HTML document specified by `$input` to XML, and returns a document node:
	• The input may either be a string or a binary item (xs:hexBinary, xs:base64Binary).
	• If the input is passed on in its binary representation, the HTML parser will try to automatically choose the correct encoding.
	The `$options` argument can be used to set TagSoup Options, which can be specified…
	• as children of an `<html:options/>` element; e.g.:
	`<html:options>`
	`  <html:key1 value='value1'/>`
	`  ...`
	`</html:options>`
	• as map, which contains all key/value pairs:
	`map { "key1": "value1", ... }`

Errors	`BXHL0001`: the input cannot be converted to XML.

Examples

Basic Example

The following query converts the specified string to an XML document node.

Query

```
html:parse("<html>")
```

Result

```
<html xmlns="http://www.w3.org/1999/xhtml"/>
```

Specifying Options

The next query creates an XML document without namespaces:

Query

```
html:parse("<a href='ok.html'/>", map { 'nons': true() })
```

Result

```
<html>
  <body>
    <a shape="rect" href="ok.html"/>
  </body>
</html>
```

Parsing Binary Input

If the input encoding is unknown, the data to be processed can be passed on in its binary representation. The HTML parser will automatically try to detect the correct encoding:

Query

```
html:parse(fetch:binary("http://en.wikipedia.org"))
```

Result

```
<html xmlns="http://www.w3.org/1999/xhtml" class="client-nojs" dir="ltr" lang="en">
  <head>
    <title>Wikipedia, the free encyclopedia</title>
    <meta charset="UTF-8"/>
    ...
```

Errors

Code	Description
BXHL0001	The input cannot be converted to XML.

Changelog

The module was introduced with Version 7.6.

Chapter 47. HTTP Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains a single function to send HTTP requests and handle HTTP responses. The function `send-request` is based on the EXPath HTTP Client Module. It gives full control over the available request and response parameters. For simple GET requests, the Fetch Module may be sufficient.

Conventions

All functions in this module are assigned to the `http://expath.org/ns/http-client` namespace, which is statically bound to the `http` prefix. All errors are assigned to the `http://expath.org/ns/error` namespace, which is statically bound to the `exerr` prefix.

Functions

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 specification. Both basic and digest authentication is supported.
Errors	`HC0001`: an HTTP error occurred.`HC0002`: error parsing the entity content as XML or HTML.`HC0003`: with a multipart response, the override-media-type must be either a multipart media type or application/octet-stream.`HC0004`: the src attribute on the body element is mutually exclusive with all other attribute (except the media-type).`HC0005`: the request element is not valid.`HC0006`: a timeout occurred waiting for the response.

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 must be contained 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>
```

The response content type can also be overwritten in order to retrieve HTML pages and other textual data as plain string (using `text/plain`) or in its binary representation (using `application/octet-stream`). With the `http:header` element, a custom user agent can be set. See the following example:

Query:

```
let $binary :=  http:send-request(
  <http:request method='get'
     override-media-type='application/octet-stream'
     href='http://www.google.com'>
    <http:header name="User-Agent" value="Opera"/>
  </http:request>
)[2]
return try {
  html:parse($binary)
} catch * {
  'Conversion to XML failed: ' || $err:description
}
```

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>
```

Errors

Code	Description
HC0001	An HTTP error occurred.
HC0002	Error parsing the entity content as XML or HTML.
HC0003	With a multipart response, the override-media-type must be either a multipart media type or application/octet-stream.
HC0004	The src attribute on the body element is mutually exclusive with all other attribute (except the media-type).
HC0005	The request element is not valid.
HC0006	A timeout occurred waiting for the response.

Changelog

Version 8.0

- Added: digest authentication

Version 7.6

- Updated: http:send-request: `HC0002` is raised if the input cannot be parsed or converted to the final data type.
- Updated: errors are using `text/plain` as media-type.

Chapter 48. Index Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for displaying information stored in the database index structures.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/index` namespace, which is statically bound to the `index` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

index:facets

Signatures	`index:facets($db as xs:string) as xs:string` `index:facets($db as xs:string, $type as xs:string) as xs:string`
Summary	Returns information about all facets and facet values of the database `$db` in document structure format.If `$type` is specified as `flat`, the function returns this information in a flat summarized version. The returned data is derived from the Path Index.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.
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($db as xs:string) as element(value)*` `index:texts($db as xs:string, $prefix as xs:string) as element(value)*` `index:texts($db as xs:string, $start as xs:string, $ascending as xs:boolean) as element(value)*`
Summary	Returns all strings stored in the Text Index of the database `$db`, along with their number of occurrences.If `$prefix` is specified, the returned entries will be refined to the ones starting with that prefix.If `$start` and `$ascending` are specified, all nodes will be returned after or before the specified start entry.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0004`: the text index is not available.

index:attributes

Signatures	`index:attributes($db as xs:string) as element(value)*` `index:attributes($db as xs:string, $prefix as xs:string) as element(value)*` `index:attributes($db as xs:string, $start as xs:string, $ascending as xs:boolean) as element(value)*`
Summary	Returns all strings stored in the Attribute Index of the database `$db`, along with their number of occurrences.If `$prefix` is specified, the returned entries will be refined to the ones starting with that prefix.If `$start` and `$ascending` are specified, all nodes will be returned after or before the specified start entry.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.`BXDB0004`: the attribute index is not available.

index:element-names

Signatures	`index:element-names($db as xs:string) as element(value)*`
Summary	Returns all element names stored in the Name Index of the database `$db`, along with their number of occurrences.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.

index:attribute-names

Signatures	`index:attribute-names($db as xs:string) as element(value)*`
Summary	Returns all attribute names stored in the Name Index of the database `$db`, along with their number of occurrences.
Errors	`BXDB0002`: The addressed database does not exist or could not be opened.

Changelog

Version 7.7

- Updated: the functions no longer accept Database Nodes as reference. Instead, the name of a database must now be specified.

Version 7.3

- Updated: index:texts, index:attributes: signature with three arguments added.

The module was introduced with Version 7.1.

Chapter 49. Inspection Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for extracting internal information about modules and functions and generating documentation.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/inspect` namespace, which is statically bound to the `inspect` prefix. xqDoc document instances are assigned to the `http://www.xqdoc.org/1.0` namespace, which is statically bound to the `xqdoc` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Reflection

inspect:functions

Signatures	`inspect:functions() as function(*)* inspect:functions($uri as xs:string) as function(*)*`
Summary	Returns function items for all user-defined functions (both public and private) that are known in the current query context. If a `$uri` is specified, the addressed file will be compiled, its functions will be added to the query context and returned to the user.
Examples	Invokes the declared functions and returns its values:

```
declare %private function local:one() { 12 };
declare %private function local:two() { 34 };
for $f in inspect:functions() return $f()
```

Compiles all functions in `code.xqm` and invokes the function named `run`:

```
let $uri := 'code.xqm'
let $name := "run"
for $f in inspect:functions($uri)
where local-name-from-QName(function-name($f)) = $name
return $f()
```

Documentation

inspect:function

Signatures	`inspect:function($function as function(*)) as element(function)`
Summary	Inspects the specified `$function` and returns an element that describes its structure. The output of this function is similar to eXist-db's inspect:inspect-function function.
Examples	The query `inspect:function(count#1)` yields:

```
<function name="count" uri="http://www.w3.org/2005/xpath-functions">
  <argument type="item()" occurrence="*"/>
  <return type="xs:integer"/>
</function>
```

The function…

```
(:~
 : This function simply returns the specified integer.
 : @param $number  number to return
 : @return         specified number
 :)
declare %private function local:same($number as xs:integer) as
 xs:integer {
  $number
};
```

…is represented by `inspect:function(local:same#1)` as…

```
<function name="local:same" uri="http://www.w3.org/2005/xquery-local-
functions">
  <argument type="xs:integer" name="number">number to return</argument>
  <annotation name="private" uri="http://www.w3.org/2012/xquery"/>
  <description>This function simply returns the specified integer.</
description>
  <return type="xs:integer">specified number</return>
</function>
```

inspect:context

Signatures	`inspect:context() as element(context)`
Summary	Generates an element that describes all variables and functions in the current query context.
Examples	Evaluate all user-defined functions with zero arguments in the query context:

```
inspect:context()/function ! function-lookup(QName(@uri, @name), 0) ! .
()
```

Return the names of all private functions in the current context:

```
for $f in inspect:context()/function
where $f/annotation/@name = 'private'
return $f/@name/string()
```

inspect:module

Signatures	`inspect:module($input as xs:string) as element(module)`
Summary	Retrieves the string from the specified `$input`, parses it as XQuery module, and generates an element that describes its structure.
Errors	`FODC0002`: the addressed resource cannot be retrieved.
Examples	An example is shown below.

inspect:xqdoc

Signatures	`inspect:xqdoc($input as xs:string) as element(xqdoc:xqdoc)`
Summary	Retrieves the string from the specified `$input`, parses it as XQuery module, and generates an xqDoc element.xqDoc provides a simple vendor neutral solution for generating a documentation from XQuery modules. The documentation conventions have been inspired by the JavaDoc standard. Documentation comments begin with `(:~` and end with `:)`, and tags start with `@`. xqDoc comments can be specified for main and library modules and variable and function declarations. We have slightly extended the xqDoc conventions to do justice to the current status of XQuery (Schema: xqdoc-1.1.30052013.xsd):

	• an `<xqdoc:annotations/>` node is added to each variable or function that uses annotations. The xqdoc:annotation child nodes may have additional `xqdoc:literal` elements with `type` attributes (xs:string, xs:integer, xs:decimal, xs:double) and values. • a single `<xqdoc:namespaces/>` node is added to the root element, which summarizes all prefixes and namespace URIs used or declared in the module. • name and type elements are added to variables
Errors	`FODC0002`: the addressed resource cannot be retrieved.
Examples	An example is shown below.

Examples

This is the `sample.xqm` library module:

```
(:~
 : This module provides some sample functions to demonstrate
 : the features of the Inspection Module.
 :
 : @author   BaseX Team
 : @see      http://docs.basex.org/wiki/XQDoc_Module
 : @version  1.0
 :)
module namespace samples = 'http://basex.org/modules/samples';

(:~ This is a sample string. :)
declare variable $samples:test-string as xs:string := 'this is a string';

(:~
 : This function simply returns the specified integer.
 : @param   $number number to return
 : @return  specified number
 :)
declare %private function samples:same($number as xs:integer) as xs:integer {
  $number
};
```

If `inspect:module('sample.xqm')` is run, the following output will be generated:

```
<module prefix="samples" uri="http://basex.org/modules/samples">
  <description>This module provides some sample functions to demonstrate
the features of the Inspection Module.</description>
  <author>BaseX Team</author>
  <see>http://docs.basex.org/wiki/XQDoc_Module</see>
  <version>1.0</version>
  <variable name="samples:test-string" uri="http://basex.org/modules/samples"
 type="xs:string">
    <description>This is a sample string.</description>
  </variable>
  <function name="samples:same" uri="http://basex.org/modules/samples">
    <argument name="number" type="xs:integer">number to return</argument>
    <annotation name="private" uri="http://www.w3.org/2012/xquery"/>
    <description>This function simply returns the specified integer.</description>
    <return type="xs:integer">specified number</return>
  </function>
</module>
```

The output looks as follows if `inspect:xqdoc('sample.xqm')` is called:

```
<xqdoc:xqdoc xmlns:xqdoc="http://www.xqdoc.org/1.0">
```

```
  <xqdoc:control>
    <xqdoc:date>2013-06-01T16:59:33.654+02:00</xqdoc:date>
    <xqdoc:version>1.1</xqdoc:version>
  </xqdoc:control>
  <xqdoc:module type="library">
    <xqdoc:uri>http://basex.org/modules/samples</xqdoc:uri>
    <xqdoc:name>sample.xqm</xqdoc:name>
    <xqdoc:comment>
      <xqdoc:description>This module provides some sample functions to demonstrate
the features of the Inspection Module.</xqdoc:description>
      <xqdoc:author>BaseX Team</xqdoc:author>
      <xqdoc:see>http://docs.basex.org/wiki/XQDoc_Module</xqdoc:see>
      <xqdoc:version>1.0</xqdoc:version>
    </xqdoc:comment>
  </xqdoc:module>
  <xqdoc:namespaces>
    <xqdoc:namespace prefix="samples" uri="http://basex.org/modules/samples"/>
  </xqdoc:namespaces>
  <xqdoc:imports/>
  <xqdoc:variables>
    <xqdoc:variable>
      <xqdoc:name>samples:test-string</xqdoc:name>
      <xqdoc:comment>
        <xqdoc:description>This is a sample string.</xqdoc:description>
      </xqdoc:comment>
      <xqdoc:type>xs:string</xqdoc:type>
    </xqdoc:variable>
  </xqdoc:variables>
  <xqdoc:functions>
    <xqdoc:function arity="1">
      <xqdoc:comment>
        <xqdoc:description>This function simply returns the specified integer.</
xqdoc:description>
        <xqdoc:param>$number number to return</xqdoc:param>
        <xqdoc:return>specified number</xqdoc:return>
      </xqdoc:comment>
      <xqdoc:name>samples:same</xqdoc:name>
      <xqdoc:annotations>
        <xqdoc:annotation name="private"/>
      </xqdoc:annotations>
      <xqdoc:signature>declare %private function samples:same($number as
 xs:integer) as xs:integer</xqdoc:signature>
      <xqdoc:parameters>
        <xqdoc:parameter>
          <xqdoc:name>number</xqdoc:name>
          <xqdoc:type>xs:integer</xqdoc:type>
        </xqdoc:parameter>
      </xqdoc:parameters>
      <xqdoc:return>
        <xqdoc:type>xs:integer</xqdoc:type>
      </xqdoc:return>
    </xqdoc:function>
  </xqdoc:functions>
</xqdoc:xqdoc>
```

Changelog

Version 7.9

- Updated: a query URI can now be specified with inspect:functions.

This module was introduced with Version 7.7.

Chapter 50. JSON Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to parse and serialize JSON documents. JSON (JavaScript Object Notation) 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 various mappings, all of which are suited to different use cases.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/json` namespace, which is statically bound to the `json` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Conversion Formats

Direct

The `direct` conversion format allows a lossless conversion from JSON to XML and back. The transformation is based on the following rules:

- The resulting document has a `<json>` root node.
- Object pairs are represented via elements. The name of a pair is rewritten to an element name:
  - Empty names are represented by a single underscore (_). Existing underscores are rewritten to two underscores (__), and characters that are not valid in element names are rewritten to an underscore and the character's four-digit Unicode.
  - If the `lax` option is set to `true`, invalid characters are simply replaced with underscores or (when invalid as first character of an element name) prefixed with an underscore. The resulting names are better readable, but cannot always be converted back to their original form.
- Array entries are also represented via elements. _ is used as element name.
- Object and array values are stored in text nodes.
- The types of values are represented via `type` attributes:
  - The existing types are *string*, *number*, *boolean*, *null*, *object*, and *array*.
  - As most values are strings, the *string* type is by default omitted.

Attributes

The `attributes` format is lossless, too. The transformation based on the following rules:

- The resulting document has a `<json>` root node.
- Object pairs are represented via `<pair>` elements. The name of a pair is stored in a `name` attribute.
- Array entries are represented via `<item>` elements.
- Object and array values are stored in text nodes.
- The types of values are represented via `type` attributes:
  - The existing types are *string*, *number*, *boolean*, *null*, *object*, and *array*.

- As most values are strings, the *string* type is by default omitted.

Map

The `map` format is lossless, too. It converts a JSON document to an XQuery map and vice versa. The conversion rules are the same as for fn:parse-json.

Basic

Introduced with Version 8.2:

The `basic` format is another lossless format. It converts a JSON document to an XML node and vice versa. The conversion rules are the same as for fn:json-to-xml.

JsonML

The `jsonml` format is designed to convert XML to JSON and back, using the JsonML dialect. JsonML allows the transformation of 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.

A little advice: in the Database Creation dialog of the GUI, if you select JSON Parsing and switch to the *Parsing* tab, you can see the effects of some of the conversion options.

Options

The following options are available (the *Direction* column indicates if an option applies to parsing, serialization, or both operations):

Option	Description	Allowed	Default	Direction
`format`	Specifies the format for converting JSON data.	`direct`, `attributes`, `jsonml`, `map`	`direct`	*parse*, *serialize*
`liberal`	Determines if minor deviations from RFC 7159 will be ignored.	`yes`, `no`	`no`	*parse*
`merge`	This option is considered when `direct` or `attributes` conversion is used: • 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 plural (*numbers*, *booleans*, *nulls*, *objects* and *arrays*), and the attribute value contains all names with that type, separated by whitespaces.	`yes`, `no`	`no`	*parse*, *serialize*
`strings`	Indicates if `type` attributes will be added for strings.	`yes`, `no`	`yes`	*parse*, *serialize*
`lax`	Specifies if a lax approach is used to convert QNames to JSON names.	`yes`, `no`	`no`	*parse*, *serialize*
`unescape`	Indicates if escaped characters are expanded (for example, `\n` becomes a single `x0A` character, while `\u20AC` becomes the character `€`).	`yes`, `no`	`yes`	*parse*
`escape`	Indicates if characters are escaped whenever the JSON syntax requires it. This option can be set to `no` if strings	`yes`, `no`	`yes`	*serialize*

	are already in escaped form and no further escaping is permitted.			
`indent`	Indicates if whitespace should be added to the output with the aim of improving human legibility. If the parameter is set as in the query prolog, it overrides the `indent` serialization parameter.	`yes`, `no`	`yes`	*serialize*

The JSON function signatures provide an `$options` argument. Options can either be specified

- as children of an `<json:options/>` element; e.g.:

```
<json:options>
  <json:format value='direct'/>
  ...
</json:options>
```

- or as map, which contains all key/value pairs:

```
map { 'format': 'direct', ... }
```

Functions

json:parse

Signatures	`json:parse($input as xs:string) as element(json)` `json:parse($input as xs:string, $options as item()) as item()`
Summary	Converts the JSON document specified by `$input` to an XML document or a map. If the input can be successfully parsed, it can be serialized back to the original JSON representation. The `$options` argument can be used to control the way the input is converted.
Errors	`BXJS0001`: the specified input cannot be parsed as JSON document.

json:serialize

Signatures	`json:serialize($input as node()) as xs:string` `json:serialize($input as node(), $options as item()) as xs:string`
Summary	Serializes the node specified by `$input` as JSON, and returns the result as `xs:string` instance. The node is expected to conform to the output created by the json:parse() function. All other items will be serialized as specified for the json output method of the official specification.Items can also be serialized as JSON if the Serialization Parameter `method` is set to `json`.The `$options` argument can be used to control the way the input is serialized.
Errors	`BXJS0002`: the specified node cannot be serialized as JSON document.

Examples

BaseX Format

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, $json, $name)
```

Example 2: Converts a simple JSON string to XML and back

Query:

```
json:parse('{}')
```

Result:

```
<json type="object"/>
```

Query:

```
(: serialize result as plain text :)
declare option output:method 'text';
json:serialize(<json type="object"/>)
```

Result:

```
{ }
```

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": "2014-02-02T11:10:27Z",
  "generator": "http://www.flickr.com/"
}')
```

Result:

```
<json type="object">
  <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>2014-02-02T11:10:27Z</modified>
  <generator>http://www.flickr.com/</generator>
</json>
```

Example 4: Converts a JSON string with different data types

Query:

```
let $options := map { 'merge': true() }
return 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
    }
  ]
}', $options)
```

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>
    <_>
      <type>home</type>
      <number>212 555-1234</number>
    </_>
    <_>
      <type>mobile</type>
      <number type="number">1327724623</number>
    </_>
  </phone>
</json>
```

JsonML Format

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(<xml/>, map { 'format': 'jsonml' })
```

Result:

```
["xml"]
```

Example 3: Converts an XML document with elements and text

Query:

```
json:serialize(doc('flickr.xml'), map { 'format': 'jsonml' })
```

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>2014-02-02T11:10:27Z</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",
    "2014-02-02T11:10:27Z"],
  ["generator",
    "http://www.flickr.com/"]]
```

Example 4: Converts a document with nested elements and attributes

Query:

```
json:serialize(doc('input.xml'), map { 'format': 'jsonml' })
```

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", {"type":"home"},
    "212 555-1234"]]
```

Errors

Code	Description
BXJS0001	The specified input cannot be parsed as JSON document.
BXJS0002	The specified node cannot be serialized as JSON document.

Changelog

Version 8.2

- Added: Conversion format `basic`.

Version 8.0

- Updated: Serialization aligned with the `json` output method of the official specification.
- Added: `liberal` option.
- Removed: `spec` option.

Version 7.8

- Removed: `json:parse-ml`, `json:serialize-ml`.
- Updated: `json:parse` now returns a document node instead of an element, or an XQuery map if `format` is set to `.map`.

Version 7.7.2

- Updated: `$options` argument added to json:parse and json:serialize.
- Updated: json:parse-ml and json:serialize-ml are now *deprecated*.

The module was introduced with Version 7.0.

Chapter 51. Map Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for manipulating maps, which will officially be introduced with XQuery 3.1. **Please note** that the functions are subject to change until the specification has reached its final stage.

Conventions

All functions in this module are assigned to the `http://www.w3.org/2005/xpath-functions/map` namespace, which is statically bound to the `map` prefix.

Functions

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:contains

Signatures	`map:contains($input as map(*), $key as xs:anyAtomicType) as xs:boolean`
Summary	Returns true if the *map* supplied as `$input` contains an entry with a key equal to the supplied value of `$key`; otherwise it returns false. No error is raised if the map contains keys that are not comparable with the supplied `$key`. If the supplied key is `xs:untypedAtomic`, it is compared as an instance of `xs:string`. If the supplied key is the `xs:float` or `xs:double` value `NaN`, the function returns true if there is an entry whose key is `NaN`, or false otherwise.
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 xs:anyAtomicType, $value as item()*) as map(*)`

Summary	Creates a new *map* containing a single entry. 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 compared as an instance of `xs:string`. If the supplied key is the `xs:float` or `xs:double` value `NaN`, the function returns the value in the entry whose key is `NaN`, or the empty sequence otherwise. The function `map:entry` is intended primarily for use in conjunction with the function `map:merge`. For example, a map containing seven entries may be constructed like this:
	`map:merge((`
	`  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:merge(for $b in //book return map:entry($b/isbn, $b))
```

Examples `map:entry("M", "Monday")` creates `map { "M": "Monday" }`.

map:for-each

Signatures `map:for-each($input as map(*), $fun as function(xs:anyAtomicType, item()*) as item()*) as item()*`

Summary Applies a function to every entry of the map `$input` and returns the results as a sequence. The function supplied as `$fun` takes two arguments. It is called supplying the key of the map entry as the first argument, and the associated value as the second argument.

Examples The following query adds the keys and values of all map entries and returns `(3,7)`:

```
map:for-each(
  map { 1: 2, 3: 4 },
  function($a, $b) { $a + $b }
)
```

map:get

Signatures `map:get($input as map(*), $key as xs:anyAtomicType) 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 `$input` that has a key equal to the supplied value of `$key`. If there is such an entry, it returns the associated value; otherwise it returns an empty sequence. No error is raised if the map contains keys that are not comparable with the supplied `$key`. If the supplied key is `xs:untypedAtomic`, it is converted to `xs:string`. If the supplied key is the `xs:float` or `xs:double` value `NaN`, the function returns an empty sequence. A return value of `()` from `map:get` could indicate that the key is present in the map with an associated value of `()`, or it could indicate that the key is not present in the map. The two cases can be distinguished by calling `map:contains`. Invoking the *map* as a function item has the same effect as calling `get`: that is, when `$input` is a map, the expression `$input($K)` is equivalent to `get($input, $K)`. Similarly, the expression `get(get(get($input, 'employee'), 'name'), 'first')` can be written as `$input('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($input 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 `$input` argument and returns the keys that are present in the map as a sequence of atomic values. The order may differ from the order in which entries were inserted in the map.

Examples	• `map:keys(map { 1: "yes", 2: "no" })` returns `(1,2)`.

map:merge

Signatures	`map:merge($input as map(*)*) as map(*)`
Summary	Constructs and returns a new map. The *map* is formed by combining the contents of the maps supplied in the `$input` argument. The 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. 2. The associated value for each such key is taken from the last map in the input sequence `$input` that contains an entry with this key. There is no requirement that the supplied input maps should have the same or compatible types. The type of a map (for example `map(xs:integer, xs:string)`) is descriptive of the entries it currently contains, but is not a constraint on how the map may be combined with other maps.
Examples	• `map:merge(())` creates an empty map. • `map:merge((map:entry(0, "no"), map:entry(1, "yes")))` creates `map { 0: "no", 1: "yes" }`. • `map:merge(($week, map { 7: "Unbekannt" }))` creates `map { 0: "Sonntag", 1: "Montag", 2: "Dienstag", 3: "Mittwoch", 4: "Donnerstag", 5: "Freitag", 6: "Samstag", 7: "Unbekannt" }`. • `map:merge(($week, map { 6: "Sonnabend" }))` creates `map { 0: "Sonntag", 1: "Montag", 2: "Dienstag", 3: "Mittwoch", 4: "Donnerstag", 5: "Freitag", 6: "Sonnabend" }`.

map:put

Signatures	`map:put($input as map(*), $key as xs:anyAtomicType, $value as item()*) as map(*)`
Summary	Creates a new *map*, containing the entries of the `$input` argument and a new entry composed by `$key` and `$value`. The semantics of this function are equivalent to `map:merge(($input, map { $key, $value }))`

map:remove

Signatures	`map:remove($input as map(*), $key as xs:anyAtomicType) as map(*)`
Summary	Constructs a new map by removing an entry from an existing map. The entries in the new map correspond to the entries of `$input`, 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 `map { 0: "Sonntag", 1: "Montag", 2: "Dienstag", 3: "Mittwoch", 5: "Freitag", 6: "Samstag" }`. • `map:remove($week, 23)` creates `map { 0: "Sonntag", 1: "Montag", 2: "Dienstag", 3: "Mittwoch", 4: "Donnerstag", 5: "Freitag", 6: "Samstag" }`.

map:size

Signatures	`map:size($input as map(*)) as xs:integer`

Summary	Returns a the number of entries in the supplied map. The function takes any *map* as its `$input` argument and returns the number of entries that are present in the map.
Examples	• `map:size(map:merge(()))` returns `0`. • `map:size(map { "true": 1, "false": 0 })` returns `2`.

map:serialize

Signatures	`map:serialize($input as map(*)) as xs:string`
Summary	This function is specific to BaseX. It returns a string representation of the supplied map. The purpose of this function is to get an insight into the structure of a map item; it cannot necessarily be used for reconstructing the original map.
Examples	• `map:serialize(map { 'A': (.1, xs:date('2001-01-01')) })` returns `{ "A": (0.1, "2001-01-01") }`.

Changelog

Version 8.0

- Added: `map:for-each`, `map:merge`, `map:put`
- Removed: support for collations (in accordance with the XQuery 3.1 spec).
- Removed: `map:new` (replaced with `map:merge`)
- Updated: aligned with latest specification: compare keys of type `xs:untypedAtomic` as `xs:string` instances, store `xs:float` or `xs:double` value `NaN`.
- Introduction on maps is now found in the article on XQuery 3.1.

Version 7.8

- Updated: map syntax `map { 'key': 'value' }`
- Added: map:serialize

Version 7.7.1

- Updated: alternative map syntax without `map` keyword and `:` as key/value delimiter (e.g.: `{ 'key': 'value' })`

Chapter 52. Math Module

Read this entry online in the BaseX Wiki.

The math XQuery Module defines functions to perform mathematical operations, such as `pi`, `asin` and `acos`. Most functions are specified in the Functions and Operators Specification of the upcoming XQuery 3.0 Recommendation, and some additional ones have been added in this module.

Conventions

All functions in this module are assigned to the `http://www.w3.org/2005/xpath-functions/math` namespace, which is statically bound to the `math` prefix.

W3 Functions

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: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($arg as xs:double?) as xs:double?`
Summary	Returns the tangent of `$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`.

Additional Functions

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: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`.

math:crc32

Signatures	`math:crc32($str as xs:string) as xs:hexBinary`
Summary	Calculates the CRC32 check sum of the given string `$str`.
Examples	• `math:crc32("")` returns `'00000000'`. • `math:crc32("BaseX")` returns `'4C06FC7F'`.

Changelog

Version 7.5

- Moved: math:random and math:uuid have been move to Random Module.

Version 7.3

- Added: math:crc32 and math:uuid have been adopted from the obsolete Utility Module.

Chapter 53. Output Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for simplifying formatted data output.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/out` namespace, which is statically bound to the `out` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

out:nl

Signatures	`out:nl() as xs:string`
Summary	Returns a single newline character (`
`).

out:tab

Signatures	`out:tab() as xs:string`
Summary	Returns a single tabulator character (` `).

out:format

Signatures	`out: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.
Examples	• `out:format("%b", true())` returns `true`. • `out:format("%06d", 256)` returns `000256`. • `out:format("%e", 1234.5678)` returns `1.234568e+03`.

Changelog

Introduced with Version 7.3. Functions have been adopted from the obsolete Utility Module.

Chapter 54. Process Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for executing system commands from XQuery.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/proc` namespace, which is statically bound to the `proc` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

proc:system

Signatures	`proc:system($cmd as xs:string) as xs:string` `proc:system($cmd as xs:string, $args as xs:string*) as xs:string` `proc:system($cmd as xs:string, $args as xs:string*, $encoding as xs:string) as xs:string`
Summary	Executes the specified command in a separate process and returns the result as string.`$cmd` is the name of the command. Arguments to the command may be specified via `$args`.The result can be explicitly converted to a specified `$encoding`. If no encoding is specified, the system's default encoding is used.
Errors	`BXPRnnnn`: If the command results in an error, an XQuery error will be raised. Its code will consist of the letters `BXPR` and four digits with the command's exit code.`BXPR9999`: the specified encoding does not exist or is not supported.
Examples	• `proc:system('date')` returns the current date on a Linux system. • The following example returns "Command not found", if the command "xyz" cannot be located or executed: `try { proc:system('xyz') } catch bxerr:BXPR0002 { 'Command not found.' }`

proc:execute

Signatures	`proc:execute($cmd as xs:string) as element(result)` `proc:execute($cmd as xs:string, $args as xs:string*) as element(result)` `proc:execute($cmd as xs:string, $args as xs:string*, $encoding as xs:string) as element(result)`
Summary	Executes the specified command in a separate process and returns the result as element.`$cmd` is the name of the command. Arguments to the command may be specified via `$args`.The result can be explicitly converted to a specified `$encoding`. If no encoding is specified, the system's default encoding is used.A result has the following structure: `<result> <output>...result output...</output> <error>...error output...</error> <code>0</code> </result>`
Errors	`BXPR9999`: the specified encoding does not exist or is not supported.

Examples

- `proc:execute('dir', '\')` returns the files of the root directory of a Windows system.
- `proc:execute('ls', ('-l', '-a'))` executes the `ls -la` command on Unix systems.

Errors

Code	Description
BXPR9999	The specified encoding does not exist or is not supported.

Changelog

The module was introduced with Version 7.3.

Chapter 55. Profiling Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains various testing, profiling and helper functions.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/prof` namespace, which is statically bound to the `prof` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

prof:time

Signatures	`prof:time($expr as item()) as item()*` `prof:time($expr as item(), $cache as xs:boolean) as item()*` `prof:time($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.
Properties	The function is *non-deterministic*: evaluation order will be preserved by the compiler.
Examples	• `prof:time("1 to 100000")` may output `25.69 ms`. • `prof:time("1 to 100000", true())` may output `208.12 ms`.

prof:mem

Signatures	`prof:mem($expr as item()) as item()*` `prof:mem($expr as item(), $cache as xs:boolean) as item()*` `prof:mem($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.
Properties	The function is *non-deterministic*: evaluation order will be preserved by the compiler.
Examples	• `prof:mem("1 to 100000")` may output `0 Bytes`. • `prof:mem("1 to 100000", true())` may output `26.678 mb`.

prof:sleep

Signatures	`prof:sleep($ms as xs:integer) as empty-sequence()`
Summary	Sleeps for the specified number of milliseconds.
Properties	The function is *non-deterministic*: evaluation order will be preserved by the compiler.

prof:human

Signatures	`prof:human($number as xs:integer) as xs:string`
Summary	Returns a human-readable representation of the specified `$number`.

Example	• `prof:human(16384)` returns `16K`.

prof:dump

Signatures	`prof:dump($expr as item()) as empty-sequence()` `prof:dump($expr as item(), $label as xs:string) as empty-sequence()`
Summary	Dumps a serialized representation of `$expr` to `STDERR`, optionally prefixed with `$label`, and returns an empty sequence. If the GUI is used, the dumped result is shown in the Info View.
Properties	In contrast to `fn:trace()`, the consumed expression will not be passed on.

prof:variables

Signatures	`prof:variables() as empty-sequence()`
Summary	Prints a list of all current local and global variable assignments to standard error or, if the GUI is used, to the Info View.As every query is optimized before being evaluated, not all of the original variables may be visible in the output. Moreover, many variables of function calls will disappear because functions are inlined. Function inlining can be turned off by setting the INLINELIMIT option to `0`.
Properties	The function is *non-deterministic*: evaluation order will be preserved by the compiler.
Examples	• `for $x in 1 to 2 return prof:variables()` will dump the values of `$x` to standard error.

prof:current-ms

Signatures	`prof:current-ms() as xs:integer`
Summary	Returns the number of milliseconds passed since 1970/01/01 UTC. The granularity of the value depends on the underlying operating system and may be larger. For example, many operating systems measure time in units of tens of milliseconds.
Properties	In contrast to `fn:current-time()`, the function is *non-deterministic*, as it returns different values every time it is called. Its evaluation order will be preserved by the compiler.

prof:current-ns

Signatures	`prof:current-ns() as xs:integer`
Summary	Returns the current value of the most precise available system timer in nanoseconds.
Properties	In contrast to `fn:current-time()`, the function is *non-deterministic*, as it returns different values every time it is called. Its evaluation order will be preserved by the compiler.

prof:void

Signatures	`prof:void($value as item()*) as empty-sequence()`
Summary	Swallows all items of the specified `$value` and returns an empty sequence. This function is helpful if some code needs to be evaluated and if the actual result is irrelevant.
Properties	The function is *non-deterministic*: evaluation order will be preserved by the compiler.
Examples	• `prof:void(fetch:binary('http://my.rest.service'))` performs an HTTP request and ignores the result.

Changelog

Version 8.1

- Added: `prof:variables`

Version 7.7

- Added: `prof:void`

Version 7.6

- Added: `prof:human`

Version 7.5

- Added: `prof:dump`, `prof:current-ms`, `prof:current-ns`

This module was introduced with Version 7.3.

Chapter 56. Random Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for computing random values. All functions except for random:seeded-double and random:seeded-integer are non-deterministic, i.#e., they return different values for each call.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/random` namespace, which is statically bound to the `random` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

random:double

Signatures	`random:double() as xs:double`
Summary	Returns a double value between 0.0 (inclusive) and 1.0 (exclusive).

random:integer

Signatures	`random:integer() as xs:integer` `random:integer($max as xs:integer) as xs:integer`
Summary	Returns an integer value, either in the whole integer range or between 0 (inclusive) and the given maximum (exclusive)
Errors	`BXRA0001`: the maximum value is out of bounds.

random:seeded-double

Signatures	`random:seeded-double($seed as xs:integer, $num as xs:integer) as xs:double*`
Summary	Returns a sequence with `$num` double values between 0.0 (inclusive) and 1.0 (exclusive). The random values are created using the initial seed given in `$seed`.

random:seeded-integer

Signatures	`random:seeded-integer($seed as xs:integer, $num as xs:integer) as xs:integer*` `random:seeded-integer($seed as xs:integer, $num as xs:integer, $max as xs:integer) as xs:integer*`
Summary	Returns a sequence with `$num` integer values, either in the whole integer range or between 0 (inclusive) and the given maximum (exclusive). The random values are created using the initial seed given in `$seed`.
Errors	`BXRA0001`: the maximum value is out of bounds.`BXRA0002`: the number of values to be returned is negative.

random:gaussian

Signatures	`random:gaussian($num as xs:integer) as xs:double*`
Summary	Returns a sequence with `$num` double values. The random values are Gaussian (i.e. normally) distributed with the mean 0.0. and the derivation 1.0.

random:uuid

Signatures	`random:uuid() as xs:string`
Summary	Creates a random universally unique identifier (UUID), represented as 128-bit value.
Examples	• `random:uuid() eq random:uuid()` will (most probably) return the boolean value `false`.

Errors

Code	Description
BXRA0001	The specified maximum value is out of bounds.
BXRA0002	The specified number of values to be returned is negative.

Changelog

Version 8.0

- Updated: random:integer, random:seeded-integer raise error for invalid input.

The module was introduced with Version 7.5. It includes some functionality which was previously located in the Math Module.

Chapter 57. Repository Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for installing, listing and deleting modules contained in the Repository.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/repo` namespace, which is statically bound to the `repo` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

repo:install

Signatures	`repo:install($path as xs:string) as empty-sequence()`
Summary	Installs a package or replaces an existing package. The parameter `$path` indicates the path to the package.
Errors	`BXRE0001`: the package does not exist.`BXRE0002`: a package uses an invalid namespace URI.`BXRE0003`: the package to be installed requires a package which is still not installed.`BXRE0004`: the package descriptor is invalid.`BXRE0005`: the module contained in the package to be installed is already installed as part of another package.`BXRE0006`: the package cannot be parsed.`BXRE0009`: the package version is not supported.`BXRE0010`: the package contains an invalid JAR descriptor.`BXRE0011`: 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 the name, suffixed with a hyphen and the package version.
Errors	`BXRE0007`: the package cannot be deleted.`BXRE0008`: another package depends on the package to be deleted.

repo:list

Signatures	`repo:list() as element(package)*`
Summary	Lists the names and versions of all currently installed packages.

Errors

Code	Description
`BXRE0001`	The addressed package does not exist.
`BXRE0002`	A package uses an invalid namespace URI.
`BXRE0003`	The package to be installed requires a package which is not installed yet.
`BXRE0004`	The package descriptor is invalid.
`BXRE0005`	The module contained in the package to be installed is already installed as part of another package.
`BXRE0006`	The package cannot be parsed.
`BXRE0007`	The package cannot be deleted.

BXRE0008	Another package depends on the package to be deleted
BXRE0009	The package version is not supported.
BXRE0010	The package contains an invalid JAR descriptor.
BXRE0011	The package contains a JAR descriptor but it cannot be read.

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.

Chapter 58. Request Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for retrieving information on an HTTP request that has triggered the query. It is mainly useful in the context of Web Applications.

The module is related to Adam Retter's EXQuery Request Module draft.

Conventions

- The `basex-api` package must be included in the classpath. This is always the case if you use one of the complete distributions (zip, exe, war) of BaseX.
- All functions are assigned to the `http://exquery.org/ns/request` namespace. The module must be imported in the query prolog:

```
import module namespace request = "http://exquery.org/ns/request";
...
```

- In this documentation, the namespace is bound to the `request` prefix.
- The following example demonstrates what components a URI may consist of (the example is derived from RFC 3986):

```
  foo://example.com:8042/over/there?name=ferret
  \_/   \_________/ \__/\_________/ \_________/
   |         |       |       |          |
 scheme   hostname  port    path      query
```

General Functions

request:method

Signatures	`request:method() as xs:string`
Summary	Returns the Method of the HTTP request.

request:attribute

Signatures	`request:attribute($name as xs:string) as xs:string`
Summary	Returns the value of an attribute of the HTTP request. If the attribute does not exist, an empty sequence is returned.
Example	• `request:attribute("javax.servlet.error.request_uri")` returns the original URI of a caught error. • `request:attribute("javax.servlet.error.message")` returns the error message of a caught error.

URI Functions

request:scheme

Signatures	`request:scheme() as xs:string`

Summary	Returns the Scheme component of the URI of an HTTP request.
Example	For the example given in the introduction, this function would return `foo`.

request:hostname

Signatures	`request:hostname() as xs:string`
Summary	Returns the Hostname component of the URI of an HTTP request.
Example	For the example given in the introduction, this function would return `example.com`.

request:port

Signatures	`request:port() as xs:integer`
Summary	Returns the Port component of the URI of an HTTP request, or a default port if it has not been explicitly specified in the URI.
Example	For the example given in the introduction, this function would return `8042`.

request:path

Signatures	`request:path() as xs:string`
Summary	Returns the Path component of the URI of an HTTP request.
Example	For the example given in the introduction, this function would return `/over/there`.

request:query

Signatures	`request:query() as xs:string?`
Summary	Returns the Query component of the URI of an HTTP request. If no query has been specified, an empty sequence is returned.
Example	For the example given in the introduction, this function would return `name=ferret`.

request:uri

Signatures	`request:uri() as xs:anyURI`
Summary	Returns the complete URI of an HTTP request as it has been specified by the client.
Example	For the example given in the introduction, this method would return `foo://example.com:8042/over/there?name=ferret`.

request:context-path

Signatures	`request:context-path() as xs:string`
Summary	Returns the context of the request. For servlets in the default (root) context, this method returns an empty string.

Connection Functions

request:address

Signatures	`request:address() as xs:string`
Summary	Returns the IP address of the server.

request:remote-hostname

Signatures	`request:remote-hostname() as xs:string`
Summary	Returns the fully qualified hostname of the client that sent the request.

request:remote-address

Signatures	`request:remote-address() as xs:string`
Summary	Returns the IP address of the client that sent the request.

request:remote-port

Signatures	`request:remote-port() as xs:string`
Summary	Returns the TCP port of the client socket that triggered the request.

Parameter Functions

request:parameter-names

Signatures	`request:parameter-names() as xs:string*`
Summary	Returns the names of all query and form field parameters available from the HTTP request. With RESTXQ, this function can be used to access parameters that have not been statically bound by %rest:query-param.
Example	For the example given in the introduction, this function would return `name`.

request:parameter

Signatures	`request:parameter($name as xs:string) as xs:string*` `request:parameter($name as xs:string, $default as xs:string) as xs:string*`
Summary	Returns the value of the named query or form field parameter in an HTTP request. If the parameter does not exist, an empty sequence or the optionally specified default value is returned instead. If both query and form field parameters with the same name exist, the form field values will be attached to the query values.
Example	For the example given in the introduction, the function call `request:parameter('name')` would return `ferret`.

Header Functions

request:header-names

Signatures	`request:header-names() as xs:string*`
Summary	Returns the names of all headers available from the HTTP request. If RESTXQ is used, this function can be used to access headers that have not been statically bound by %rest:header-param.

request:header

Signatures	`request:header($name as xs:string) as xs:string?` `request:header($name as xs:string, $default as xs:string) as xs:string`
Summary	Returns the value of the named header in an HTTP request. If the header does not exist, an empty sequence or the optionally specified default value is returned instead.

Cookie Functions

request:cookie-names

Signatures	`request:cookie-names() as xs:string*`
Summary	Returns the names of all cookies in the HTTP headers available from the HTTP request. If RESTXQ is used, this function can be used to access cookies that have not been statically bound by %rest:cookie-param.

request:cookie

Signatures	`request:cookie($name as xs:string) as xs:string*` `request:cookie($name as xs:string, $default as xs:string) as xs:string`
Summary	Returns the value of the named Cookie in an HTTP request. If there is no such cookie, an empty sequence or the optionally specified default value is returned instead.

Changelog

Version 7.9

- Updated: The returned values of request:parameter-names, request:parameter now also include form field parameters.

Version 7.8

- Added: request:context-path

Version 7.7

- Added: request:attribute

This module was introduced with Version 7.5.

Chapter 59. RESTXQ Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains helper functions for the RESTXQ API, some of which are defined in the RESTXQ Draft.

Conventions

- The `basex-api` package must be included in the classpath. This is always the case if you use one of the complete distributions (zip, exe, war) of BaseX.
- All functions are assigned to the `http://exquery.org/ns/restxq` namespace. The module must be imported in the query prolog:

```
import module namespace rest = "http://exquery.org/ns/restxq";
...
```

- In this documentation, the namespace is bound to the `rest` prefix, and the `http://wadl.dev.java.net/2009/02` namespace is bound to the `wadl` prefix.
- If any of the functions is called outside the servlet context, the error `BXSE0003`: is raised.

General Functions

rest:base-uri

Signatures	`rest:base-uri() as xs:anyURI`
Summary	This function returns the implementation-defined base URI of the resource function.

rest:uri

Signatures	`rest:uri() as xs:anyURI`
Summary	This function returns the complete URI that addresses the Resource Function. This is the result of rest:base-uri appended with the path from the path annotation of the resource function.

rest:wadl

Signatures	`rest:wadl() as element(wadl:application)`
Summary	This (unofficial) function returns a WADL description of all available REST services.

Changelog

This module was introduced with Version 7.7.

Chapter 60. Session Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for accessing and modifying server-side session information. This module is mainly useful in the context of Web Applications.

Conventions

- The `basex-api` package must be included in the classpath. This is always the case if you use one of the complete distributions (zip, exe, war) of BaseX.
- All functions are assigned to the `http://basex.org/modules/session` namespace. The module must be imported in the query prolog:

```
import module namespace session = "http://basex.org/modules/session";
...
```

- In this documentation, the namespace is bound to the `session` prefix.
- Errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.
- If any of the functions is called outside the servlet context, the error `BXSE0003`: is raised.
- As sessions are side-effecting operations, all functions are flagged as *non-deterministic*. This means that the functions will not be reordered by the compiler.

Functions

session:id

Signatures	`session:id() as xs:string`
Summary	Returns the session ID of a servlet request.

Examples	Running the server-side XQuery file `id.xq` via `http://localhost:8984/id.xq`:
	`import module namespace session = "http://basex.org/modules/session";`
	`'Session ID: ' || session:id()`

session:created

Signatures	`session:created() as xs:dateTime`
Summary	Returns the creation time of a session.

session:accessed

Signatures	`session:accessed() as xs:dateTime`
Summary	Returns the last access time of a session.

session:names

Signatures	`session:names() as xs:string*`
Summary	Returns the names of all variables bound to the current session.

Examples	Running the server-side XQuery file `names.xq` via `http://localhost:8984/names.xq`:

```
import module namespace session = "http://basex.org/modules/session";
session:names() ! element variable { . }
```

session:get

Signatures	`session:get($key as xs:string) as item()* session:get($key as xs:string, $default as item()*) as item()*`
Summary	Returns the value of a variable bound to the current session. If the key is unknown, an empty sequence or the optionally specified default value is returned instead.
Errors	`BXSE0002`: the value of a session variable could not be retrieved.
Examples	Running the server-side XQuery file `get.xq` via `http://localhost:8984/get.xq?key=user`:

```
import module namespace session = "http://basex.org/modules/session";
'Value of ' || $key || ': ' || session:get($key)
```

session:set

Signatures	`session:set($key as xs:string, $value as item()*) as empty-sequence()`
Summary	Binds the specified key/value pair to a session.
Errors	`BXSE0001`: a function item was specified as value of a session variable.
Examples	Running the server-side XQuery file `set.xq` via `http://localhost:8984/set.xq?key=user&value=john`:

```
import module namespace session = "http://basex.org/modules/session";
session:set($key, $value), 'Variable was set.'
```

session:delete

Signatures	`session:delete($key as xs:string) as empty-sequence()`
Summary	Deletes a session variable.
Examples	Running the server-side XQuery file `delete.xq` via `http://localhost:8984/delete.xq?key=user`:

```
import module namespace session = "http://basex.org/modules/session";
session:delete($key), 'Variable was deleted.'
```

session:close

Signatures	`session:close() as empty-sequence()`
Summary	Unregisters a session and all data associated with it.

Errors

Code	Description
`BXSE0001`	A function item was specified as value of a session attribute.
`BXSE0002`	An error occurred while retrieving the value of a session attribute.

`BXSE0003` A function was called outside the servlet context.

Changelog

Version 8.0

- Updated: allow sequences as session values

This module was introduced with Version 7.5.

Chapter 61. Sessions Module

Read this entry online in the BaseX Wiki.

This XQuery Module can only be called from users with *Admin* permissions. It contains functions for accessing and modifying all registered server-side sessions. This module is mainly useful in the context of Web Applications.

Conventions

- The `basex-api` package must be included in the classpath. This is always the case if you use one of the complete distributions (zip, exe, war) of BaseX.
- All functions are assigned to the `http://basex.org/modules/sessions` namespace. The module must be imported in the query prolog:

```
import module namespace sessions = "http://basex.org/modules/sessions";
...
```

- In this documentation, the namespace is bound to the `sessions` prefix.
- Errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.
- If any of the functions is called outside the servlet context, the error `BXSE0003`: is raised.
- As sessions are side-effecting operations, all functions are flagged as *non-deterministic*. This means that the functions will not be reordered by the compiler.

Functions

sessions:ids

Signatures	`sessions:ids() as xs:string`
Summary	Returns the IDs of all registered sessions.

sessions:created

Signatures	`sessions:created($id as xs:string) as xs:dateTime`
Summary	Returns the creation time of the session specified by `$id`.

sessions:accessed

Signatures	`sessions:accessed($id as xs:string) as xs:dateTime`
Summary	Returns the last access time of the session specified by `$id`.

sessions:names

Signatures	`sessions:names($id as xs:string) as xs:string*`
Summary	Returns the names of all variables bound to the session specified by `$id`.

sessions:get

Signatures	`sessions:get($id as xs:string, $key as xs:string) as xs:string?` `sessions:get($id as xs:string, $key as xs:string, $default as xs:string) as xs:string`

Summary	Returns the value of a variable bound to the session specified by `$id`. If the variable does not exist, an empty sequence or the optionally specified default value is returned instead.
Errors	`BXSE0002`: the value of a session variable could not be retrieved.

sessions:set

Signatures	`sessions:set($id as xs:string, $key as xs:string, $value as xs:string) as empty-sequence()`
Summary	Assigns a value to a variable bound to the session specified by `$id`.
Errors	`BXSE0001`: a function item was specified as value of a session variable.

sessions:delete

Signatures	`sessions:delete($id as xs:string, $key as xs:string) as empty-sequence()`
Summary	Deletes a variable bound to the session specified by `$id`.

sessions:close

Signatures	`sessions:close($id as xs:string) as empty-sequence()`
Summary	Unregisters the session specified by `$id`.

Errors

Code	Description
`BXSE0001`	A function item was specified as value of a session attribute.
`BXSE0002`	An error occurred while retrieving the value of a session attribute.
`BXSE0003`	A function was called outside the servlet context.
`BXSE0004`	The specified session was not found.

Changelog

This module was introduced with Version 7.5.

Chapter 62. SQL Module

Read this entry online in the BaseX Wiki.

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.

This module uses JDBC to connect to a SQL server. Hence, your JDBC driver will need to be added to the classpath, too. If you work with the full distributions of BaseX, you can copy the driver into the `lib` directory. To connect to MySQL, for example, download the Connector/J Driver and extract the archive into this directory.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/sql` namespace, which is statically bound to the `sql` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

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	`BXSQ0007`: 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 `$password` are specified, they are used as credentials for connecting to the database. The `$options` parameter can be used to set connection options, which can either be specified
	• as children of an `<sql:options/>` element, e.g.:
	`<sql:options>`
	`  <sql:autocommit value='true'/>`
	`  ...`
	`</sql:options>`
	• as map, which contains all key/value pairs:
	`map { "autocommit": true(), ... }`

Errors	`BXSQ0001`: 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, $query as xs:string) as element()*`
Summary	This function executes a query or update statement. The parameter `$connection` specifies a connection handle. The parameter `$query` is a string representing an SQL statement.
Errors	`BXSQ0001`: an SQL exception occurs, e.g. not existing relation is retrieved. `BXSQ0002`: a wrong connection handle is passed.

sql:execute-prepared

Signatures	`sql:execute-prepared($id as xs:integer, $params as element(sql:parameters)) as element()*`
Summary	This function executes a prepared statement. The parameter `$id` specifies a prepared statement handle. The optional parameter `$params` is an element `<sql:parameters/>` representing the parameters for a prepared statement along with their types and values. 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	`BXSQ0001`: an SQL exception occurs, e.g. not existing relation is retrieved. `BXSQ0002`: a wrong prepared statement handle is passed. `BXSQ0003`: the number of `<sql:parameter/>` elements in `<sql:parameters/>` differs from the number of placeholders in the prepared statement. `BXSQ0004`: the type of a parameter for a prepared statement is not specified. `BXSQ0005`: an attribute different from `type` and `null` is set for a `<sql:parameter/>` element. `BXSQ0006`: 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	`BXSQ0001`: an SQL exception occurs. `BXSQ0002`: a wrong connection handle is passed.

sql:commit

Signatures	`sql:commit($connection as xs:integer) as empty-sequence()`
Summary	This function commits the changes made to a relational database. `$connection` specifies the connection handle.
Errors	`BXSQ0001`: an SQL exception occurs. `BXSQ0002`: a wrong connection handle is passed.

sql:rollback

Signatures	`sql:rollback($connection as xs:integer) as empty-sequence()`
Summary	This function rolls back the changes made to a relational database. `$connection` specifies the connection handle.

Errors	`BXSQ0001`: an SQL exception occurs. `BXSQ0002`: a wrong connection handle is passed.

sql:close

Signatures	`sql:close($connection as xs:integer) as empty-sequence()`
Summary	This function closes a connection to a relational database. `$connection` specifies the connection handle.
Errors	`BXSQ0001`: an SQL exception occurs. `BXSQ0002`: 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">2014-10-10 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-prepared($prep, $params)
```

SQLite

The following expression demonstrates how SQLite can be addressed using the Xerial SQLite JDBC driver:

```
(: Initialize driver :)
sql:init("org.sqlite.JDBC"),
(: Establish a connection :)
let $conn := sql:connect("jdbc:sqlite:database.db")
return (
  (: Create a new table :)
  sql:execute($conn, "drop table if exists person"),
  sql:execute($conn, "create table person (id integer, name string)"),
  (: Run 10 updates :)
  for $i in 1 to 10
  let $q := "insert into person values(" || $i || ", '" || $i || "')"
  return sql:execute($conn, $q),
  (: Return table contents :)
  sql:execute($conn, "select * from person"),
  sql:close($conn)
)
```

Errors

Code	Description
BXSQ0001	An SQL exception occurred (e.g.: a non-existing relation is retrieved).
BXSQ0002	A wrong connection handle or prepared statement handle is passed.
BXSQ0003	The number of `<sql:parameter/>` elements in `<sql:parameters/>` differs from the number of placeholders in the prepared statement.
BXSQ0004	The type of a parameter for a prepared statement is not specified.
BXSQ0005	An attribute different from `type` and `null` is set for a `<sql:parameter/>` element.
BXSQ0006	A parameter is from type date, time or timestamp and its value is in an invalid format.
BXSQ0007	A specified database driver class is not found.

Changelog

Version 7.5

- Updated: prepared statements are now executed via sql:execute-prepared

The module was introduced with Version 7.0.

Chapter 63. Streaming Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for handling *streamable* items.

In contrast to standard XQuery items, a streamable item contains only a reference to the actual data. The data itself will be retrieved if it is requested by an expression, or if the item is to be serialized. Hence, a streamable item only uses a few bytes, and no additional memory is occupied during serialization.

The following BaseX functions return streamable items:

- Streamable Base64 binaries:
  - `db:retrieve`
  - `fetch:binary`
  - `file:read-binary`
- Streamable strings:
  - `fetch:text`
  - `file:read-text`

Some functions are capable of consuming items in a *streamable* fashion: data will never be cached, but instead passed on to another target (file, the calling expression, etc.). The following streaming functions are currently available:

- `convert:binary-to-bytes`
- `db:store`
- `file:write-binary`
- `file:write-text`

The XQuery expression below serves as an example on how large files can be downloaded and written to a file with constant memory consumption:

```
file:write-binary('output.data', fetch:binary('http://files.basex.org/xml/
xmark111mb.zip'))
```

Conventions

All functions in this module are assigned to the `http://basex.org/modules/stream` namespace, which is statically bound to the `stream` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

stream:materialize

Signatures	`stream:materialize($value as item()*) as item()*`
Summary	Returns a materialized instance of the specified `$value`: • if an item is streamable, its value will be retrieved, and a new item containing the value will be returned.

	• other, non-streamable items will simply be passed through. Materialization is advisable if a value is to be processed more than once, and is expensive to retrieve. It is get mandatory whenever a value is invalidated before it is requested (see the example below).

Example	In the following example, a file will be deleted before its content is returned. To avoid a "file not found" error, the content will first be materialized:
	
	`let $file := 'data.txt'`
	`let $data := stream:materialize(file:read-text($file))`
	`return (file:delete($file), $data)`

stream:is-streamable

Signatures	`stream:is-streamable($item as item()) as item()`
Summary	Checks whether the specified `$item` is streamable.

Changelog

Version 8.0

- Update: stream:materialize extended to sequences.

This module was introduced with Version 7.7.

Chapter 64. Unit Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains annotations and functions for performing XQUnit tests.

Introduction

The more complex a software application grows, the more error-prone it gets. This is why testing frameworks have been developed, which provide a standardized, automatized way for testing software. The XUnit frameworks (such as SUnit or JUnit) allow testing of atomic unit of a program, such as single functions and algorithms.

This module borrows heavily from the existing frameworks: it provides various annotations for testing XQuery functions. Unit functions are provided to assert the validity of arbitrary conditions expressed in XQuery and to raise errors whenever a condition is not satisfied. Some additional functions exist to run all unit tests of the current module or a set of specified library modules.

Usage

Tests are started via the TEST command. It compiles all XQuery modules in a given file or directory and runs all functions that are annotated with `%unit:test`. A test report is generated and returned, which resembles the format returned by other xUnit testing frameworks, such as the Maven Surefire Plugin (see below).

Conventions

All annotations, functions and errors in this module are assigned to the `http://basex.org/modules/unit` namespace, which is statically bound to the `unit` prefix.

Annotations

%unit:test

Syntax	`%unit:test` `%unit:test("expected", CODE)`
Summary	With this annotation, a function can be marked as unit test. It will be evaluated if a test report is created for the module in which this function is located.`error` can be supplied as additional string argument. It is followed by `CODE`, which must be a valid EQName string. If the function expression does not raise that error, the test will fail.

Examples	• The following test does will be successful, as it does nothing (and, hence, nothing wrong):
	`declare %unit:test function local:void() { () };`
	• The following test will be successful, as the function body will raise `err:XPTY0004`:
	`declare %unit:test('expected', "err:XPTY0004") function local:add() { 123 + 'strings and integers cannot to added' };`

%unit:before

Syntax	`%unit:before` `%unit:before(FUNCTION)`
Summary	A function decorated with this annotation will be evaluated **before each** unit test.`FUNCTION` can be supplied as additional argument. It must be a valid EQName string. If specified, the function will only be evaluated before a function with the given name is tested. This extension is e. g. helpful if the results of updates need to be tested.

Examples

- The first function will be evaluated before the actual test:

```
declare %updating %unit:before("local:check") function local:before-
check() {
  db:create('test-db')
};
declare %updating %unit:test function local:check() {
  unit:assert(db:exists('test-db'))
};
```

%unit:after

Syntax	`%unit:after` `%unit:after(FUNCTION)`
Summary	A function decorated with this annotation will be evaluated **after each** unit test.`FUNCTION` can be supplied as additional argument. It must be a valid EQName string. If specified, the function will only be evaluated after a function with the given name is tested.

%unit:before-module

Syntax	`%unit:before-module`
Summary	If a function is decorated with this annotation, it will be evaluated **before all** unit tests in the current module.

%unit:after-module

Syntax	`%unit:after-module`
Summary	If a function is decorated with this annotation, it will be evaluated **after all** unit tests in the current module.

%unit:ignore

Syntax	`%unit:ignore` `%unit:ignore(MESSAGE)`
Summary	If a function is decorated with this annotation, it will temporarily be ignored by the test suite runner.

Functions

unit:assert

Signatures	`unit:assert($test as item()*) as empty-sequence()` `unit:assert($test as item()*, $info as item()) as empty-sequence()`
Summary	Asserts that the effective boolean value of the specified `$test` is true and returns an empty sequence. Otherwise, raises an error. The *effective boolean value* of an expression can be explicitly computed by using the `fn:boolean` function.The default failure message can be overridden with the `$info` argument.
Errors	`UNIT0001`: the assertion failed, or an error was raised.

unit:assert-equals

Signatures	`unit:assert-equals($returned as item()*, $expected as item()*) as empty-sequence()` `unit:assert-equals($returned as item()*, $expected as item()*, $info as item()) as empty-sequence()`
Summary	Asserts that the specified arguments are equal according to the rules of the `fn:deep-equals` function. Otherwise, raises an error.The default failure message can be overridden with the `$info` argument.

Errors	`UNIT0001`: the assertion failed, or an error was raised.

unit:fail

Signatures	`unit:fail() as empty-sequence()` `unit:fail($info as item()) as empty-sequence()`
Summary	Raises a unit error. The default failure message can be overridden with the `$info` argument.
Errors	`UNIT0001`: default error raised by this function.

Example

The following XQUnit module `tests.xqm` contains all available unit annotations:

Query

```
module namespace test = 'http://basex.org/modules/xqunit-tests';

(:~ Initializing function, which is called once before all tests. :)
declare %unit:before-module function test:before-all-tests() {
  ()
};

(:~ Initializing function, which is called once after all tests. :)
declare %unit:after-module function test:after-all-tests() {
  ()
};

(:~ Initializing function, which is called before each test. :)
declare %unit:before function test:before() {
  ()
};

(:~ Initializing function, which is called after each test. :)
declare %unit:after function test:after() {
  ()
};

(:~ Function demonstrating a successful test. :)
declare %unit:test function test:assert-success() {
  unit:assert(<a/>)
};

(:~ Function demonstrating a failure using unit:assert. :)
declare %unit:test function test:assert-failure() {
  unit:assert((), 'Empty sequence.')
};

(:~ Function demonstrating a failure using unit:assert-equals. :)
declare %unit:test function test:assert-equals-failure() {
  unit:assert-equals(4 + 5, 6)
};

(:~ Function demonstrating an unexpected success. :)
declare %unit:test("expected", "err:FORG0001") function test:unexpected-success() {
  ()
};

(:~ Function demonstrating an expected failure. :)
declare %unit:test("expected", "err:FORG0001") function test:expected-failure() {
  1 + <a/>
};
```

```
(:~ Function demonstrating the creation of a failure. :)
declare %unit:test function test:failure() {
  unit:fail("Failure!")
};

(:~ Function demonstrating an error. :)
declare %unit:test function test:error() {
  1 + <a/>
};

(:~ Skipping a test. :)
declare %unit:test %unit:ignore("Skipped!") function test:skipped() {
  ()
};
```

By running `TEST tests.xqm`, the following report will be generated (timings may differ):

Result

```
<testsuites time="PT0.256S">
  <testsuite name="file:///C:/Users/user/Desktop/test.xqm" time="PT0.212S"
 tests="8" failures="4" errors="1" skipped="1">
    <testcase name="assert-success" time="PT0.016S"/>
    <testcase name="assert-failure" time="PT0.005S">
      <failure line="30" column="15">
        <info>Empty sequence.</info>
      </failure>
    </testcase>
    <testcase name="assert-equals-failure" time="PT0.006S">
      <failure line="35" column="22">
        <returned item="1" type="xs:integer">9</returned>
        <expected item="1" type="xs:integer">6</expected>
        <info>Item 1: 6 expected, 9 returned.</info>
      </failure>
    </testcase>
    <testcase name="unexpected-success" time="PT0.006S">
      <failure>
        <expected>FORG0001</expected>
      </failure>
    </testcase>
    <testcase name="expected-failure" time="PT0.004S"/>
    <testcase name="failure" time="PT0.004S">
      <failure line="50" column="13">
        <info>Failure!</info>
      </failure>
    </testcase>
    <testcase name="error" time="PT0.004S">
      <error line="55" column="6" type="FORG0001">
        <info>Cannot cast to xs:double: "".</info>
      </error>
    </testcase>
    <testcase name="skipped" skipped="Skipped!" time="PT0S"/>
  </testsuite>
</testsuites>
```

Errors

Code	Description
`UNIT0001`	An assertion failed, or an error was raised.
`UNIT0002`	A test function must have no arguments.

`UNIT0003`	A test function is not public.
`UNIT0004`	An annotation was declared twice.
`UNIT0005`	An annotation has invalid arguments.

Changelog

Version 8.0.2

- Updated: (expected) errors are compared by QNames instead of local names (including namespaces).

Version 8.0

- Deleted: `UNIT0006` (ignore results returned by functions).
- Added: unit:fail, 0-argument signature.
- Updated: the info argument of functions can now be an arbitrary item.
- Updated: infos are now represented in an `info` child element.
- Updated: unit:before and unit:after can be extended by a filter argument.

Version 7.9

- Added: TEST command
- Removed: unit:test, unit:test-uris

Version 7.8

- Added: unit:assert-equals
- Updated: enhanced test report output

This module was introduced with Version 7.7.

Chapter 65. User Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for creating and administering database users. The User Management article gives more information on database users and permissions.

Conventions

All functions in this module and errors are assigned to the `http://basex.org/modules/user` namespace, which is statically bound to the `user` prefix.

Functions

user:current

Signatures	`user:current() as xs:string`
Summary	Returns the name of the currently logged in user.
Examples	If the GUI or the standalone mode is used, `user:current()` always returns `admin`.

user:list

Signatures	`user:list() as xs:string*`
Summary	Returns the names of all registered users.
Examples	• `user:list()` returns all registered users.

user:list-details

Signatures	`user:list-details() as element(user)*` `user:list-details($name as xs:string) as element(user)*`
Summary	Returns an element sequence, containing all registered users and their permissions.In addition to the SHOW USERS command, encoded password strings and database permissions will be output. A user `$name` can be specified to filter the results in advance.
Examples	• By default, `user:list-details()` returns the following output:

```
<user name="admin" permission="admin">
  <password algorithm="digest">
    <hash>304bdfb0383c16f070a897fc1eb25cb4</hash>
  </password>
  <password algorithm="salted-sha256">
    <salt>871602799292195</salt>

 <hash>a065ca66fa3d6da5762c227587f1c8258c6dc08ee867e44a605a72da115dcb41</
hash>
  </password>
</user>
```

Errors	`unknown`: The specified user name is unknown.

user:exists

Signatures	`user:exists($name as xs:string) as xs:boolean`
Summary	Checks if a user with the specified `$name` exists.

Examples	• `user:exists('admin')` will always yield true.
Errors	`name`: The specified user name is invalid.

user:create

Signatures	`user:create($name as xs:string, $password as xs:string) as empty-sequence()` `user:create($name as xs:string, $password as xs:string, $permission as xs:string) as empty-sequence()`
Summary	Creates a new user with the specified `$name` and `$password`. The default permission `none` can be overwritten with the `$permission` argument. Existing users will be overwritten.
Examples	• `user:create('John', '7e$j#!1', 'admin')` creates a new user 'John' with admin permissions.
Errors	`name`: The specified user name is invalid.`permission`: The specified permission is invalid.`admin`: The "admin" user cannot be modified.`logged-in`: The specified user is currently logged in.`update`: The operation can only be performed once per user or database pattern.

user:grant

Signatures	`user:grant($name as xs:string, $permission as xs:string) as empty-sequence()` `user:grant($name as xs:string, $permission as xs:string, $pattern as xs:string) as empty-sequence()`
Summary	Grants the specified `$permission` to a user with the specified `$name`. If a glob `$pattern` is specified, the permission will only be applied to databases matching that pattern.
Examples	• `user:grant('John', 'create')` grants create permissions to the user 'John'. • `user:grant('John', 'write', 'unit*')` allows John to write to all databases starting with the letters 'unit'.
Errors	`unknown`: The specified user name is unknown.`name`: The specified user name is invalid.`pattern`: The specified database pattern is invalid.`permission`: The specified permission is invalid.`admin`: The "admin" user cannot be modified.`local`: A local permission can only be 'none', 'read' or 'write'.`logged-in`: The specified user is currently logged in.`update`: The operation can only be performed once per user or database pattern.

user:drop

Signatures	`user:drop($name as xs:string) as empty-sequence()` `user:drop($name as xs:string, $pattern as xs:string) as empty-sequence()`
Summary	Drops a user with the specified `$name`. If a glob `$pattern` is specified, only the database pattern will be dropped.
Examples	• `user:drop('John')` drops the user 'John'. • `user:grant('John', 'unit*')` removes the 'unit*' database pattern. If John accesses any of these database, his global permission will be checked again.
Errors	`unknown`: The specified user name is unknown.`name`: The specified user name is invalid.`pattern`: The specified database pattern is invalid.`admin`: The "admin" user cannot be modified.`logged-in`: The specified user is currently logged in.`update`: The operation can only be performed once per user or database pattern.`conflict`: A user cannot be both altered and dropped.

user:alter

Signatures	`user:alter($name as xs:string, $newname as xs:string) as empty-sequence()`

Summary	Renames a user with the specified `$name` to `$newname`.
Examples	• `user:rename('John', 'Jack')` renames the user 'John' to 'Jack'.
Errors	`unknown`: The specified user name is unknown.`name`: The specified user name is invalid.`admin`: The "admin" user cannot be modified.`logged-in`: The specified user is currently logged in.`update`: The operation can only be performed once per user or database pattern.`conflict`: A user cannot be both altered and dropped.

user:password

Signatures	`user:password($name as xs:string, $password as xs:string) as empty-sequence()`
Summary	Changes the `password` of a user with the specified `$name`.
Examples	• `user:password('John', )` *assigns user 'John' an empty password string.*
Errors	`unknown`: The specified user name is unknown.`name`: The specified user name is invalid.`update`: The operation can only be performed once per user or database pattern.

Errors

Code	Description
`name`	The specified user name is invalid.
`pattern`	The specified database name is invalid.
`permission`	The specified permission is invalid.
`unknown`	The specified user does not exist.
`admin`	The "admin" user cannot be modified.
`equal`	Name of old and new user is equal.
`local`	A local permission can only be 'none', 'read' or 'write'.
`logged-in`	The specified user is currently logged in.
`update`	The operation can only be performed once per user or database pattern.
`conflict`	A user cannot be both altered and dropped.

Changelog

Version 8.1

- Added: user:current.

The Module was introduced with Version 8.0.

Chapter 66. Validation Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to perform validations against XML Schema and Document Type Declarations. By default, this module uses Java's standard validators. As an alternative, Saxon XSLT Processor is used if (`saxon9he.jar`, `saxon9pe.jar` or `saxon9ee.jar`) is added to the classpath.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/validate` namespace, which is statically bound to the `validate` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

validate:xsd

Signatures	`validate:xsd($input as item()) as empty-sequence()` `validate:xsd($input as item(), $schema as item()) as empty-sequence()`
Summary	Validates the document specified by `$input`. Both `$input` and `$schema` can be specified as: • `xs:string` , containing the path to the resource, • `xs:string` , containing the resource in its string representation, or • `node()` , containing the resource itself. `$schema` can be used to specify the schema for validation. If no schema is given, `$input` is required to contain an `xsi:(noNamespace)schemaLocation` attribute as defined in W3C XML Schema.
Errors	`BXVA0001`: the validation fails.`BXVA0002`: the validation process cannot be started.
Examples	• `validate:xsd('doc.xml', 'doc.xsd')` validates the document `doc.xml` against the specified schema `doc.xsd`. • The following example demonstrates how a document can be validated against a schema without resorting to local or remote URIs:

```
let $doc := <simple:root xmlns:simple='http://basex.org/simple'/>
let $schema :=
  <xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'
 targetNamespace='http://basex.org/simple'>
    <xs:element name='root'/>
  </xs:schema>
return validate:xsd($doc, $schema)
```

validate:xsd-info

Signatures	`validate:xsd-info($input as item()) as xs:string*` `validate:xsd-info($input as item(), $schema as item()) as xs:string*`
Summary	Validates the document specified by `$input` and returns warning, errors and fatal errors in a string sequence. `$input` and `$schema` can be specified as: • `xs:string` , containing the path to the resource, • `xs:string` , containing the resource in its string representation, or

	• `node()` , containing the resource itself. `$schema` can be used to specify the schema for validation. If no schema is given, `$input` is required to contain an `xsi:(noNamespace)schemaLocation` attribute as defined in W3C XML Schema.
Errors	`BXVA0002`: the validation process cannot be started.

validate:dtd

Signatures	`validate:dtd($input as item()) as empty-sequence()` `validate:dtd($input as item(), $dtd as xs:string) as empty-sequence()`
Summary	Validates the document specified by `$input`. `$input` can be specified as: • an `xs:string`, containing the path to the resource, • an `xs:string`, containing the resource in its string representation, or • a `node()`, containing the resource itself. `$schema` can be used to specify the DTD for validation. If no DTD is given, `$input` is required to contain a DTD doctype declaration.
Errors	`BXVA0001`: the validation fails.`BXVA0002`: the validation process cannot be started.
Examples	• `validate:dtd('doc.xml', 'doc.dtd')` validates the document `doc.xml` against the specified DTD file `doc.dtd`. • The following example validates an invalid document against a DTD, which is specified as string: `try { let $doc := <invalid/> let $dtd := '<!ELEMENT root (#PCDATA)>' return validate:dtd($doc, $dtd) } catch BXVA0001 { 'DTD Validation failed.' }`

validate:dtd-info

Signatures	`validate:dtd-info($input as item()) as xs:string*` `validate:dtd-info($input as item(), $dtd as xs:string) as xs:string*`
Summary	Validates the document specified by `$input` and returns warning, errors and fatal errors in a string sequence. `$input` can be specified as: • `xs:string` , containing the path to the resource, • `xs:string` , containing the resource in its string representation, or • `node()` , containing the resource itself. `$schema` can be used to specify the DTD for validation. If no DTD is given, `$input` is required to contain a DTD doctype declaration.
Errors	`BXVA0002`: the validation process cannot be started.

Errors

Code	Description

BXVA0001	The document cannot be validated against the specified DTD or XML Schema.
BXVA0002	The validation cannot be started.

Changelog

Version 7.6

- Added: validate:xsd-info, validate:dtd-info

The module was introduced with Version 7.3.

Chapter 67. Web Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides convenience functions for building web applications with RESTXQ.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/web` namespace, which is statically bound to the `web` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

web:content-type

Signatures	`web:content-type($path as xs:string) as xs:string`
Summary	Returns the content type of a path by analyzing its file suffix. `application/octet-stream` is returned if the file suffix is unknown.
Examples	• `web:content-type("sample.mp3")` returns `audio/mpeg`

web:create-url

Signatures	`web:create-url($url as xs:string, $parameters as map(*)) as xs:string`
Summary	Creates a new URL from the specified `$url` string and the `$parameters` specified in a map. The keys and and values of the map entries will be converted to strings, URL-encoded (see web:encode-url), and appended to the url as query parameters. If a map entry has more than a single item, all of them will be appended as single parameters.
Examples	• `web:create-url('http://find.me', map { 'q': 'dog' })` returns `http://find.me?q=dog` • `web:create-url('search', map { 'year': (2000,2001), 'title':() })` returns `search?year=2000&year=2001`

web:encode-url

Signatures	`web:encode-url($string as xs:string) as xs:string`
Summary	Encodes a string to a URL. Spaces are rewritten to `+`; `*`, `-`, `.` and _ are adopted; and all other non-ASCII characters and special characters are percent-encoded.
Examples	• `web:encode-url("this is a test!.html")` returns `this+is+a+test%21.html`.

web:decode-url

Signatures	`web:decode-url($string as xs:string) as xs:string`
Summary	Decodes a URL to the original string. Percent-encoded characters are decoded to their UTF8 codepoints, and `+` characters are rewritten to spaces.
Examples	• `web:decode-url("%E6%97%A5%E6%9C%AC%E8%AA%9E")` returns ###.
Errors	`BXWE0001`: Specified URI is invalid.
Errors	`BXWE0002`: Specified URI contains invalid XML characters.

web:redirect

Signatures	`web:redirect($location as xs:string) as element(rest:response)` `web:redirect($location as xs:string, $parameters as map(*)) as element(rest:response)`
Summary	Creates a RESTXQ redirection to the specified location. The returned response will only work if no other items are returned by the RESTXQ function.If `$parameters` are specified, they will be appended as query parameters to the URL as described for web:create-url.
Examples	The query `web:redirect('/a/b')` returns the following response element:

```
<rest:response xmlns:rest="http://exquery.org/ns/restxq">
  <http:response xmlns:http="http://expath.org/ns/http-client"
 status="302">
    <http:header name="location" value="/a/b"/>
  </http:response>
</rest:response>
```

web:response-header

Signatures	`web:response-header() as element(rest:response)` `web:response-header($headers as map(*)) as element(rest:response)` `web:response-header($headers as map(*), $output as map(*)) as element(rest:response)`
Summary	Creates a RESTXQ response header with a default caching directive and default serialization parameters. Header options can be supplied via the `$headers` argument. Empty string values can be specified to invalidate default values. By default, the following header options will be returned: • `Cache-Control:max-age=3600,public` Serialization parameters can be supplied via the `$output` argument. Empty string values can be specified to invalidate default values. By default, the following serialization parameters will be returned: • `media-type:application/octet-stream` • `method:raw`
Examples	• The function call `web:response-header()` returns the following response element:

```
<rest:response xmlns:rest="http://exquery.org/ns/restxq">
  <http:response xmlns:http="http://expath.org/ns/http-client">
    <http:header name="Cache-Control" value="max-age=3600,public"/>
  </http:response>
  <output:serialization-parameters xmlns:output="http://www.w3.org/2010/
xslt-xquery-serialization">
    <output:media-type value="application/octet-stream"/>
    <output:method value="raw"/>
  </output:serialization-parameters>
</rest:response>
```

• If the following RESTXQ function is called by a web browser…

```
declare %rest:path('media/{$file}') function local:get($file) {
  let $path := 'path/to/' || $file
  return (
    web:response-header(map { 'media-type': web:content-type($path) }),
    file:read-binary($path)
  )
```

```
};
```

…a file resource with the correct content-type will be returned to user (provided that it exists in the web server's file system).

Errors

Code	Description
BXWE0001	Specified URL is invalid.
BXWE0002	Specified URL contains invalid XML characters.

Changelog

Version 8.2

- Added: web:encode-url, web:decode-url

The module was introduced with Version 8.1.

Chapter 68. XQuery Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for evaluating XQuery strings and modules at runtime.

Conventions

All functions in this module are assigned to the `http://basex.org/modules/xquery` namespace, which is statically bound to the `xquery` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

xquery:eval

Signatures	`xquery:eval($query as xs:string) as item()*` `xquery:eval($query as xs:string, $bindings as map(*)) as item()*` `xquery:eval($query as xs:string, $bindings as map(*), $options as item()) as item()`

Summary	Evaluates the supplied `$query` string as XQuery expression and returns the resulting items.The evaluated query has its own query context. If a returned node is stored in a database, a main-memory copy will be returned as result, because the referenced database is closed after query execution and will not be accessible anymore.Variables and context items can be declared via `$bindings`. The specified keys must be QNames or strings:
	• If a key is a QName, it will be directly adopted as variable name.
	• It a key is a string, it may be prefixed with a dollar sign. Namespace can be specified using the Clark Notation.
	• If the specified string is empty, the value will be bound to the context item.
	The `$options` parameter contains evaluation options, which can either be specified
	• as children of an `<xquery:options/>` element:
	`<xquery:options>`
	`  <xquery:permission value="none"/>`
	`</xquery:options>`
	• as map, which contains all key/value pairs:
	`map { "permission": "none" }`
	The following options are available:
	• `permission` : the query will be evaluated with the specified permissions (see User Management).
	• `timeout` : query execution will be interrupted after the specified number of seconds.
	• `memory` : query execution will be interrupted if the specified number of megabytes will be exceeded. This check works best if only one process is running at the same time.

Errors	`BXXQ0001`: the query contains updating expressions.`BXXQ0003`: insufficient permissions for evaluating the query.`BXXQ0004`: query execution exceeded timeout or memory constraints.`FOTY0013`: the expression yields function items.Any other error that may occur while evaluating the query.

Examples

- `xquery:eval("1+3")` returns `4`.
- You can bind the context and e.g. operate on a certain database only:

```
xquery:eval("//country", map { '': db:open('factbook') })
```

- The following expressions use strings as keys. All of them return 'XML':

```
xquery:eval(".", map { '': 'XML' }),

xquery:eval("declare variable $xml external; $xml", map { 'xml':
 'XML' }),

xquery:eval(
  "declare namespace pref='URI';
   declare variable $pref:xml external;
   $pref:xml",
  map { '{URI}xml': 'XML' }
)
```

- The following expressions use QNames as keys. All of them return 'XML':

```
declare namespace pref = 'URI';

xquery:eval("declare variable $xml external; $xml", map
 { xs:QName('xml'): 'XML' }),

let $query := "declare namespace pref='URI';
               declare variable $pref:xml external;
               $pref:xml"
let $vars := map { xs:QName('pref:xml'): 'XML' }
return xquery:eval($query, $vars)
```

xquery:update

Signatures	`xquery:update($query as xs:string) as item()*` `xquery:update($query as xs:string, $bindings as map(*)) as item()*` `xquery:update($query as xs:string, $bindings as map(*), $options as item()) as item()`
Summary	Evaluates `$query` as updating XQuery expression at runtime.All updates will be added to the Pending Update List of the main query and performed after the evaluation of the main query.
Errors	`BXXQ0002`: the query contains no updating expressions.`BXXQ0003`: insufficient permissions for evaluating the query.`BXXQ0004`: query execution exceeded timeout or memory constraints.`FOTY0013`: the expression yields function items.Any other error that may occur while evaluating the query.

xquery:parse

Signatures	`xquery:parse($query as xs:string) as item()*` `xquery:parse($query as xs:string, $options as item()) as item()`
Summary	Parses the specified `$query` string as XQuery module and returns information on the resulting query plan (please note that the naming of the expressions in the query plan may change over time). The `$options` parameters can be specified in two ways: • as children of an `<xquery:options/>` element: `<xquery:options>`

```
  <xquery:compile value="true"/>
</xquery:options>
```

- as map, which contains all key/value pairs:

```
map { "compile": true() }
```

The following options are available:

- `compile` : additionally compiles the query after parsing it. By default, this option is `false`.
- `plan` : returns an XML representation of the internal query plan. By default, this option is `true`.

Errors Any error that may occur while parsing the query.

Examples

- `xquery:parse("1 + 3")` returns:

```
<MainModule updating="false">
  <QueryPlan compiled="false">
    <Arith op="+">
      <Int value="1" type="xs:integer"/>
      <Int value="3" type="xs:integer"/>
    </Arith>
  </QueryPlan>
</MainModule>
```

xquery:invoke

Signatures `xquery:invoke($uri as xs:string) as item()*` `xquery:invoke($uri as xs:string, $bindings as map(*)) as item()*` `xquery:invoke($uri as xs:string, $bindings as map(*), $options as item()) as item()*`

Summary Opens `$uri` as file, evaluates it as XQuery expression at runtime, and returns the resulting items. Database nodes in the result will be copied and returned instead.The semantics of the `$bindings` and `$options` parameters is the same as for xquery:eval.

Errors `BXXQ0001`: the expression contains updating expressions.`BXXQ0003`: insufficient permissions for evaluating the query.`BXXQ0004`: query execution exceeded timeout.`FOTY0013`: the expression yields function items.Any other error that may occur while evaluating the query.

xquery:type

Signatures `xquery: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.

Errors

Code	Description
`BXXQ0001`	The specified query contains updating expressions.
`BXXQ0002`	The specified query contains no updating expressions.
`BXXQ0003`	Insufficient permissions for evaluating the query.
`BXXQ0004`	Query execution exceeded timeout.

Changelog

Version 8.0

- Added: xquery:update, xquery:parse
- Deleted: xquery:evaluate (opened databases will now be closed by main query)

Version 7.8.2

- Added: `$options` argument

Version 7.8

- Added: xquery:evaluate
- Updated: used variables must be explicitly declared in the query string.

This module was introduced with Version 7.3. Functions have been adopted from the obsolete Utility Module.

Chapter 69. XSLT Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions and variables to perform XSLT transformations. 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 (`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();
```

Conventions

All functions in this module are assigned to the `http://basex.org/modules/xslt` namespace, which is statically bound to the `xslt` prefix. All errors are assigned to the `http://basex.org/errors` namespace, which is statically bound to the `bxerr` prefix.

Functions

xslt:processor

Signatures	`xslt:processor() as xs:string`
Summary	Returns 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	Returns 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 `$stylesheet`, and returns the result as node. `$input` and `$stylesheet` can be specified as • `xs:string` , containing the path to the document, • `xs:string` , containing the document in its string representation, or • `node()` , containing the document itself. The `$params` argument can be used to bind variables to a stylesheet, which can either be specified • as children of an `<xslt:parameters/>` element; e.g.:

```
<xslt:parameters>
  <xslt:key1 value='value1'/>
  ...
</xslt:parameters>
```

- or as map, which contains all key/value pairs:

```
map { "key1": "value1", ... }
```

Note that only strings are supported when using Saxon (XSLT 2.0).

Error	`BXSL0001`: an error occurred during the transformation process.

xslt:transform-text

Signatures	`xslt:transform-text($input as item(), $stylesheet as item()) as xs:string` `xslt:transform-text($input as item(), $stylesheet as item(), $params as item()) as xs:string`
Summary	Transforms the document specified by `$input`, using the XSLT template specified by `$stylesheet`, and returns the result as string. The parameters are the same as described for xslt:transform.
Error	`BXSL0001`: an error occurred during the transformation process.

Examples

Example 1: Basic XSL transformation with dummy document and without parameters

Query:

```
xslt:transform-text(<dummy/>, 'basic.xslt')
```

basic.xslt

```
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:template match="/">123</xsl:template>
</xsl:stylesheet>
```

Result:

```
123
```

Example 2: XSLT transformation of an input document

Query:

```
(: Outputs the result as html. :)
declare option output:method 'html';
(: Turn whitespace chopping off. :)
declare option db:chop 'no';

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 :=
  <xsl:stylesheet version='2.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output method='xml'/>
    <xsl:template match="/">
<html>
  <body>
    <div>
      <xsl:for-each select='books/book'>
      • <b><xsl:apply-templates select='title'/></b>: <xsl:value-of
 select='author'/><br/>
      </xsl:for-each>
    </div>
  </body>
</html>
    </xsl:template>
  </xsl:stylesheet>

return xslt:transform($in, $style)
```

Result:

```
<html>
  <body>
    <div>
      • <b>XSLT Programmer’s Reference</b>: Michael H. Kay<br>
      • <b>XSLT</b>: Doug Tidwell<br>
    </div>
  </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.xslt

```
<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>
```

Errors

Code	Description
BXSL0001	An error occurred during the transformation process.

Changelog

Version 7.6

- Added: xslt:transform-text
- Updated: xslt:transform returned error code

Version 7.3

- Updated: $xslt:processor → xslt:processor, $xslt:version → xslt:version

Chapter 70. ZIP Module

Read this entry online in the BaseX Wiki.

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. The module is based on the EXPath ZIP Module. It may soon be replaced by the Archive Module.

Conventions

All functions in this module are assigned to the `http://expath.org/ns/zip` namespace, which is statically bound to the `zip` prefix. All errors are assigned to the `http://expath.org/ns/error` namespace, which is statically bound to the `experr` prefix.

Functions

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	`ZIP0001`: the specified path does not exist.`ZIP0003`: 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	`ZIP0001`: the specified path does not exist.`ZIP0003`: 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`: the addressed file is not well-formed.`ZIP0001`: the specified path does not exist.`ZIP0003`: 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 is found in the classpath.
Errors	`FODC0006`: the addressed file is not well-formed, or cannot be converted to correct XML.`ZIP0001`: the specified path does not exist.`ZIP0003`: the operation fails for some other reason.

zip:entries

Signatures	`zip:entries($uri as xs:string) as element(zip:file)`
Summary	Generates an ZIP XML Representation of the hierarchical structure of the ZIP file located at `$uri` and returns it as an element node. The file contents are not returned by this function.
Errors	`ZIP0001`: the specified path does not exist.`ZIP0003`: 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.
Errors	`ZIP0001`: an addressed file does not exist.`ZIP0002`: entries in the ZIP archive description are unknown, missing, or invalid.`ZIP0003`: the operation fails for some other reason.`Serialization Errors`: 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), $output as xs:string) as empty-sequence()`
Summary	Updates an existing ZIP archive or creates a modifed copy, based on the characteristics described by `$zip`, the ZIP XML Representation. The `$output` argument is the URI where the modified ZIP file is copied to.
Errors	`ZIP0001`: an addressed file does not exist.`ZIP0002`: entries in the ZIP archive description are unknown, missing, or invalid.`ZIP0003`: the operation fails for some other reason.`Serialization Errors`: 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')`

Errors

Code	Description
`ZIP0001`	A specified path does not exist.
`ZIP0002`	Entries in the ZIP archive description are unknown, missing, or invalid.
`ZIP0003`	An operation fails for some other reason.

Part VII. Developing

Chapter 71. Developing

Read this entry online in the BaseX Wiki.

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

- Eclipse : 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
- Contribute : Collection of small projects to contribute to the BaseX Project

JavaDoc

The project's JavaDoc can be explored online.

Web Application

- RESTXQ : Write web services with XQuery
- REST : Access and update databases via HTTP requests
- WebDAV : Access databases from your filesystem
- XForms : Build browser forms with XML technologies

APIs

- Clients : Communicate with BaseX using C#, PHP, Python, Perl, C, ...
- Java Examples : Code examples for developing with BaseX
- XQJ API , implemented by Charles Foster
- XQuery for Scala API , based on XQJ and written by Dino Fancellu

Chapter 72. Developing with Eclipse

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to get the BaseX sources compiled and running on your system.

Another article in the documentation describes how to use BaseX as a query processor in Eclipse.

Prerequisites

BaseX is developed with the Eclipse environment (other IDEs like IntelliJ IDEA can be used as well). The Eclipse IDE for Java Developers includes the EGit plugin (for Git) and the m2e plugin (for Maven).

Other Eclipse plugins we use are:

Name	Description	Update URL
eclipse-cs	Enforces Checkstyle coding standards.	`http://eclipse-cs.sf.net/update/`
FindBugs	Analyze project at byte code level	`http://findbugs.cs.umd.edu/eclipse`
Core Tools	Find dead (unreferenced) code.	`http://eclipse.org/eclipse/platform-core/updates`
EclEmma	Code coverage tool.	`http://update.eclemma.org/`

Check Out

Our Git Tutorial explains how BaseX can be checked out from the GitHub Repository and embedded in Eclipse with EGit. The article also demonstrates how git can be used on command-line.

The basex repository contains the following sub-directories:

1. `basex-core` is the main project
2. `basex-api` contains the BaseX APIs (XML:DB, bindings in other languages) and HTTP Services (REST, RESTXQ, WebDAV)
3. `basex-examples` includes some examples code for BaseX
4. `basex-tests` contains several unit and stress tests

If the problems view shows a list of warning, you may need to switch to Java 6 (*Windows* → *Preferences* → *Installed JREs*).

With the Maven plugin from Eclipse, it sometimes requires several attempts to get all dependencies updated. This loop can be avoided if the sources are precompiled via Maven on command-line.

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*

Alternative

You may as well use the standalone version of Maven to compile and run the project, use other IDEs such as IntelliJ IDEA.

Chapter 73. Git

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to use git to manage the BaseX sources.

Using Git to contribute to BaseX

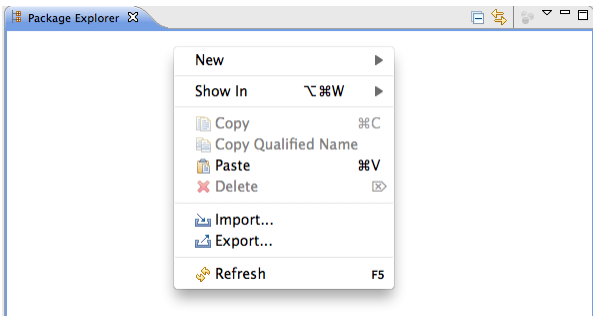
Our team uses git and GitHub to manage the source code. All team members have read+write access to the repository, and external contributors are welcome to fork the project.

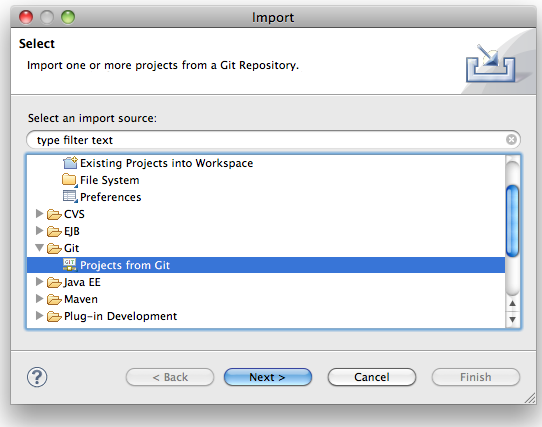
Git makes it easy to retain a full copy of the repository for yourself. To get started and running, simply *fork* BaseX:

1. Head over to https://github.com and create an account
2. Fork https://github.com/BaseXdb/basex, so you have a version on your own
3. The forked project can then be cloned on your local machine, and changes can be pushed back to your remote repository
4. Open Eclipse
5. Install egit (Eclipse: *Help* → *Marketplace* → Search for *egit* **or** get it from http://www.eclipse.org/egit/)

Using Git & Eclipse

To clone the project from within Eclipse, you may need to install EGit first (Eclipse: *Help* → *Marketplace* → Search for *egit* **or** get it from http://www.eclipse.org/egit/).



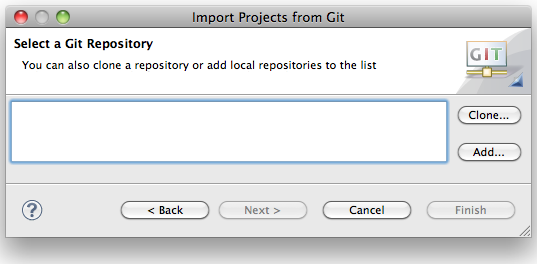


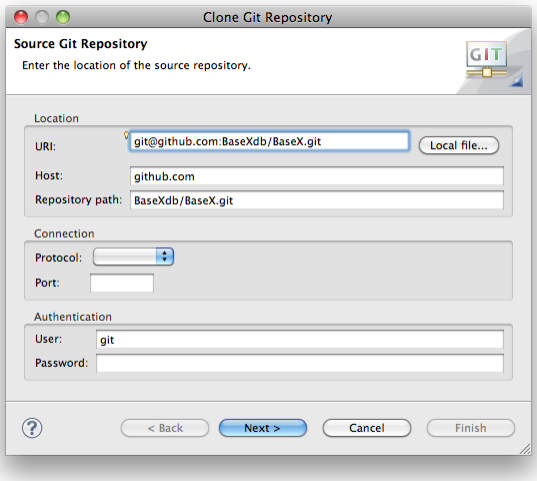
Clone

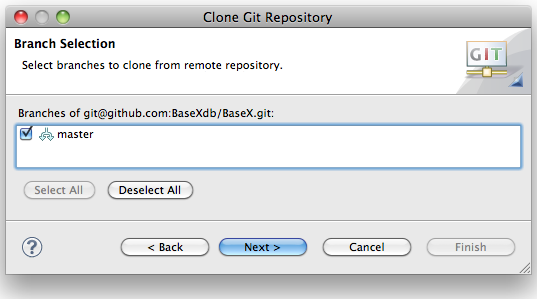
- In the **Package Explorer** to the left use right-click and choose Import...
- Select "**Projects from Git**" and click Next >
- Click "**Clone...**" to create a local copy of the remote repository. This copy will include the full project history
- Copy & Paste the GitHub URI in the Location field. 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. The read-only URI of the repository is `https://github.com/BaseXdb/basex.git`.
- 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**".

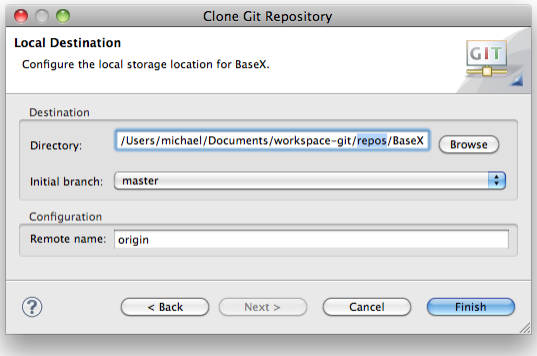
Create the project

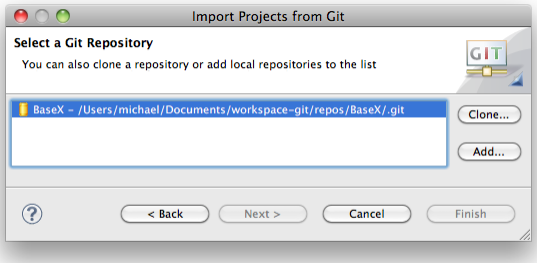
- Select our newly cloned repository and click Next
- Select "**Import Existing Projects**" and depending on your Eclipse version enable automatic sharing. More











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 library which is, however, reported 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 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 (your github user name will be referenced as $username)
2. Set up SSH access to GitHub as described here
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)

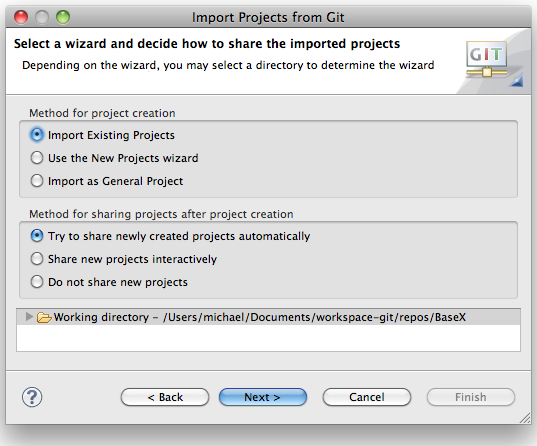
Clone Your Personal Repository

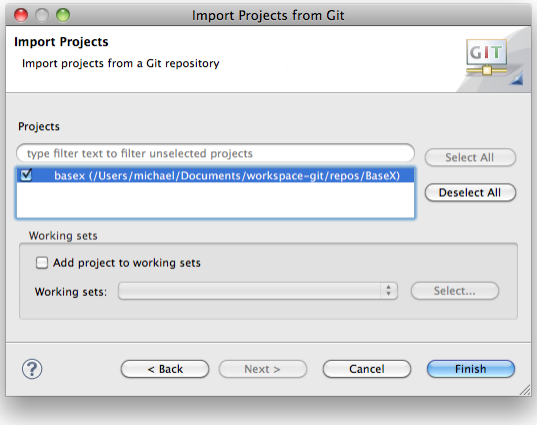
```
$ git clone git@github.com:$username/
$project.git
Cloning into $project...
Enter passphrase for key '/home/
user/.ssh/id_rsa':
...

$ ls -d -1 $PWD/*
/home/user/myprojects/$project
```

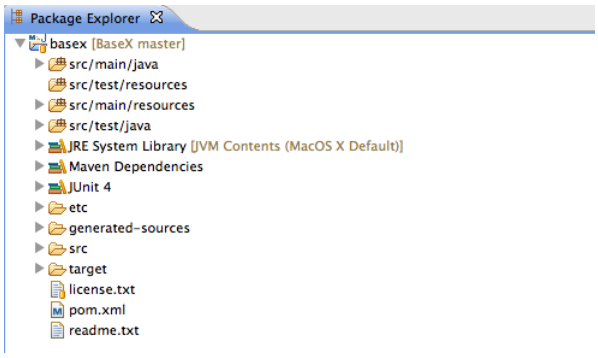
Note that git automatically creates a directory where the repository content will be checked out.

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 0fde1fb] 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 0fde1fb] 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.

Developing a new feature or bug fix using git

It is always a good idea to create a new branch for a new feature or a big fix you are working on. So first, let's make sure you have the most up-to-date source code. We assume, that you added BaseX as upstream repository as described above and you are currently in the *master* branch:

```
$ git pull upstream master
```

Now, we create a new branch, based on the master branch

```
$ git checkout -b new-feature
Switched to a new branch 'new-feature'
```

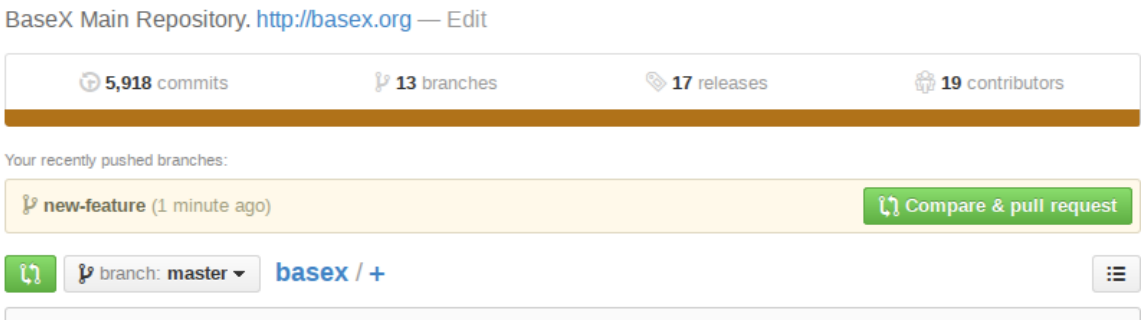
Your are now automatically switched to the *new-feature* branch. Now you can make all your changes in one or several commits. You can commit all changes using

```
$ git commit -a
```

Now, you want to push these changes to the repository on GitHub. Remember, that up to now your changes just reside on your local drive, so now you want to push it to your remote fork of BaseX. Simply do:

```
$ git push origin new-featuCounting objects: 318, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (107/107), done.
Writing objects: 100% (154/154), 22.96 KiB | 0 bytes/s, done.
Total 154 (delta 93), reused 81 (delta 26)
To git@github.com:$username/basex.git
 * [new branch]      new-feature -> new-featurere
```

You can now use your web browser and go to your fork of BaseX. You will see the following message:



You can now click the "Compare & pull request" button. You can now review the changes you are going to push.

Please review them carefully. Also, please give a meaningful comment so we can quickly determine what your changes are doing. After clicking the "Create Pull request" button you are done and we will review your changes and either merge the pull request or get back to you.

Need help using git?

Installing

For information on how to install git on various platforms please refer to: GitHub: git Installation Guide

Documentation

- Comprehensive Getting Starting Guide on GitHub
- The git book
- Gitcasts.com – Video Guides

Chapter 74. Maven

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It demonstrates how Maven is used to compile and run BaseX, and embed it into other projects.

Using Maven

If you have cloned our repository 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.
- `mvn package` : JAR archives are created in the `target` class directory, and all relevant libraries are created in the `lib` directory. Packaging is useful if you want to use the start scripts.
- `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` you can skip the JUnit tests and speed up packaging. You may as well use Eclipse and m2eclipse to compile the BaseX sources.

There are several alternatives for starting BaseX:

- type in `java -cp target/classes org.basex.BaseX` in the `basex-core` directory to start BaseX on the command-line mode,
- type in `mvn jetty:run` in the `basex-api` directory to start BaseX with Jetty and the HTTP servers,
- run one of the Start Scripts contained in the `etc` directory

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>7.6</version>
</dependency>
```

APIs and Services

...including APIs and the REST, RESTXQ and WebDAV services:

```
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
```

```
  <version>7.6</version>
</dependency>
```

XQJ API

The XQJ API is hosted at http://xqj.net:

```
<repository>
  <id>xqj</id>
  <name>XQJ Maven Repository</name>
  <url>http://xqj.net/maven</url>
</repository>
...
<dependency>
  <groupId>net.xqj</groupId>
  <artifactId>basex-xqj</artifactId>
  <version>1.2.0</version>
</dependency>
<dependency>
  <groupId>com.xqj2</groupId>
  <artifactId>xqj2</artifactId>
  <version>0.1.0</version>
</dependency>
<dependency>
  <groupId>javax.xml.xquery</groupId>
  <artifactId>xqj-api</artifactId>
  <version>1.0</version>
</dependency>
```

Chapter 75. Releases

Read this entry online in the BaseX Wiki.

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

Chapter 76. Translations

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to translate BaseX into other (natural) languages.

Thanks to the following contributors, BaseX is currently available in 10 languages:

- **Dutch** : Huib Verweij
- **English** : BaseX Team
- **French** : Maud Ingarao
- **German** : BaseX Team
- **Hungarian** : Kiss-Kálmán Dániel
- **Indonesian** : Andria Arisal
- **Italian** : Massimo Franceschet
- **Japanese** : Toshio HIRAI and Kazuo KASHIMA
- **Mongolian** : Tuguldur Jamiyansharav
- **Romanian** : Adrian Berila
- **Russian** : Oleksandr Shpak and Max Shamaev
- **Spanish** : Carlos Marcos

It is easy to translate BaseX into your native language! This is how you can proceed:

Working with the sources

If you have downloaded all BaseX sources via Eclipse or Git, you may proceed as follows:

All language files are placed in the `src/main/resources/lang` directory of the main project:

1. Create a copy of an existing translation file (e.g., `English.lang`) and rename it to your target language (e.g. `Hawaiian.lang`)
2. Enter your name and contact information in the second line
3. If you are using Eclipse, refresh the project (via *Project* → *Refresh*); if you are using Maven, type in `mvn compile`. Your new language file will be automatically detected.
4. Start the BaseX GUI, choose your language via *Options* → *Preferences...* and close the GUI
5. Translate the texts in your language file and restart BaseX in order to see the changes
6. Repeat the last step if you want to revise your translations

Updating BaseX.jar

You may directly add new languages to the JAR file. JAR files are nothing else than ZIP archives, and all language files are placed in the `lang` directory int the JAR file:

1. Unzip an existing translation file (e.g., `English.lang`) and rename it to your target language (e.g. `Hawaiian.lang`)

2. Enter your name and contact information in the second line and translate the texts

3. Update your JAR file by copying the translated file into the zipped `lang` directory. Your new language file will be automatically detected.

4. Start BaseX.jar, choose your language via *Options* → *Preferences...* and restart BaseX to see the changes

You may also change the language in the `.basex` configuration file, which is placed in your home directory. In order to see where the all text keys are used within BaseX, you may temporarily set the LANGKEY option to `true`.

Part VIII. HTTP Services

Chapter 77. RESTXQ

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the RESTXQ API of BaseX.

RESTXQ, introduced by Adam Retter, is an 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: 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.

Please note that BaseX provides various extensions to the official draft of the specification:

- Multipart types are supported, including `multipart/form-data`
- A `%rest:error` annotation can be used to catch XQuery errors
- Servlet errors can be redirected to other RESTXQ pages
- A RESTXQ Module provides some helper functions
- Parameters are implicitly cast to the type of the function argument
- The Path Annotation can contain regular expressions
- Quality factors in the Accept header will be evaluated
- `%input` annotations, support for input-specific content-type parameters

Introduction

The RESTXQ service is accessible via `http://localhost:8984/`.

All RESTXQ annotations are assigned to the `http://exquery.org/ns/restxq` namespace, which is statically bound to the `rest` prefix. A *Resource Function* is an XQuery function that has been marked up with RESTXQ annotations. When an HTTP request comes in, a resource function will be invoked that matches the constraints indicated by its annotations.

Whenever a RESTXQ URL is requested, the RESTXQPATH module directory and its sub-directories will be parsed for functions with RESTXQ annotations in library modules (detected by the extension `.xqm`) and main modules (detected by `.xq`). In main expressions, the main module will never be evaluated. All modules will be cached and parsed again when their timestamp changes.

A first RESTXQ function is shown below:

```
module namespace page = 'http://basex.org/examples/web-page';

declare %rest:path("hello/{$who}") %rest:GET function page:hello($who) {
  <response>
    <title>Hello { $who }!</title>
  </response>
};
```

If the URI http://localhost:8984/hello/World is accessed, the result will be:

```
<response>
  <title>Hello World!</title>
</response>
```

The next function demonstrates a POST request:

```
declare
  %rest:path("/form")
  %rest:POST
  %rest:form-param("message","{$message}", "(no message)")
  %rest:header-param("User-Agent", "{$agent}")
  function page:hello-postman(
    $message as xs:string,
    $agent   as xs:string*)
    as element(response)
{
  <response type='form'>
    <message>{ $message }</message>
    <user-agent>{ $agent }</user-agent>
  </response>
};
```

If you post something (e.g. using curl or the embedded form at http://localhost:8984/)...

```
curl -i -X POST --data "message='CONTENT'" http://localhost:8984/form
```

...you will receive something similar to the following result:

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 107
Server: Jetty(8.1.11.v20130520)
```

```
<response type="form">
  <message>'CONTENT'</message>
  <user-agent>curl/7.31.0</user-agent>
</response>
```

Request

This section shows how annotations are used to handle and process HTTP requests.

Constraints

Constraints restrict the HTTP requests that a resource function may process.

Paths

A resource function must have a single *Path Annotation* with a single string as argument. The function will be called if a URL matches the path segments and templates of the argument. *Path templates* contain variables in curly brackets, and map the corresponding segments of the request path to the arguments of the resource function. The first slash in the path is optional.

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 page:test($with, $variable as xs:integer) { ... };
```

Variables can be enhanced by regular expressions:

```
(: Matches all paths with "app" as first, a number as second, and "order" as third
 segment :)
declare %rest:path("app/{$code=[0-9]+}/order")
  function page:order($full-path) { ... };

(: Matches all other all paths starting with "app/" :)
declare %rest:path("app/{$path=.+}")
  function page:others($path) { ... };
```

Content Negotiation

Two following annotations can be used to restrict functions to specific content types:

- **HTTP Content Types** : a function will only be invoked if the HTTP `Content-Type` header of the request matches one of the given mime types. Example:

```
%rest:consumes("application/xml", "text/xml")
```

- **HTTP Accept** : a function will only be invoked if the HTTP `Accept` header of the request matches one of the defined mime types. Example:

```
%rest:produces("application/atom+xml")
```

By default, both mime types are `*/*`. Quality factors supplied by a client will also be considered in the path selection process. If a client supplies the following accept header…

```
*/*;q=0.5,text/html;q=1.0
```

…and if two RESTXQ functions exist with the same `path` annotation and the `produces` annotations `*/*` and `text/html`, respectively, the function with the second annotation will be called, because the quality factor for `text/html` documents is higher than the one for arbitrary other mime types.

Note that this annotation will *not* affect the content-type of the HTTP response. Instead, you will need to add a `%output:media-type` annotation.

HTTP Methods

Default Methods

The HTTP method annotations are equivalent to all HTTP request methods except TRACE and CONNECT. Zero or more methods may be used on a function; if none is specified, the function will be invoked for each method.

The following function will be called if GET or POST is used as request method:

```
declare %rest:GET %rest:POST %rest:path("/post")
  function page:post() { "This was a GET or POST request" };
```

The POST and PUT annotations may optionally take a string literal in order to map the HTTP request body to a function argument. Once again, the target variable must be embraced by curly brackets:

```
declare %rest:PUT("{$body}") %rest:path("/put")
  function page:put($body) { "Request body: " || $body };
```

Custom Methods

Custom HTTP methods can be specified with the `%rest:method` annotation:

```
declare %rest:method("RETRIEVE")
  function page:retrieve() { "RETRIEVE was specified as request method." };
```

Content Types

The body of a POST or PUT request will be converted to an XQuery item. Conversion can be controlled by specifying a content type. It can be further influenced by specifying additional content-type parameters:

Content-Type	Parameters (`;name=value`)	Type of resulting XQuery item
`text/xml`, `application/xml`		`document-node()`
`text/*`		`xs:string`
`application/json`	JSON Options	`document-node()`
`text/html`	HTML Options	`document-node()`
`text/comma-separated-values`	CSV Options	`document-node()`
others		`xs:base64Binary`
`multipart/*`		sequence (see next paragraph)

For example, if `application/json;lax=yes` is specified as content type, the input will be transformed to JSON, and the lax QName conversion rules will be applied, as described in the JSON Module.

Input options

Conversion options for JSON, CSV and HTML can also be specified via annotations using the `input` prefix. The following function treats the first line of the textual input as CSV header:

```
declare
  %rest:path("/store.csv")
  %rest:POST("{$csv}")
  %input:csv("header=true")
  function page:store-csv($csv as document-node())
{
  "Number of rows: " || count($csv/csv/record)
};
```

Multipart Types

The single parts of a multipart message are represented as a sequence, and each part is converted to an XQuery item as described in the last paragraph.

A function that is capable of handling multipart types is identical to other RESTXQ functions:

```
declare
  %rest:path("/multipart")
  %rest:POST("{$data}")
  %rest:consumes("multipart/mixed") (: optional :)
  function page:multipart($data as item()*)
{
  "Number of items: " || count($data)
};
```

Parameters

The following annotations can be used to bind request values to function arguments. Values will implicitly be cast to the type of the argument.

Query Parameters

The value of the *first parameter*, if found in the query component, will be assigned to the variable specified as *second parameter*. If no value is specified in the HTTP request, all additional parameters will be bound to the variable (if no additional parameter is given, an empty sequence will be bound):

```
declare
  %rest:path("/params")
  %rest:query-param("id", "{$id}")
  %rest:query-param("add", "{$add}", 42, 43, 44)
  function page:params($id as xs:string?, $add as xs:integer+)
{
  <result id="{ $id }" sum="{ sum($add) }"/>
};
```

HTML Form Fields

Form parameters are specified the same way as query parameters. Their values are the result of HTML forms submitted with the content type `application/x-www-form-urlencoded`.

```
%rest:form-param("parameter", "{$value}", "default")
```

File Uploads

Files can be uploaded to the server by using the content type `multipart/form-data` (the HTML5 `multiple` attribute enables the upload of multiple files):

```
<form action="/upload" method="POST" enctype="multipart/form-data">
  <input type="file" name="files"  multiple="multiple"/>
  <input type="submit"/>
</form>
```

The file contents are placed in a map, with the filename serving as key. The following example shows how uploaded files can be stored in a temporary directory:

```
declare
  %rest:POST
  %rest:path("/upload")
  %rest:form-param("files", "{$files}")
  function page:upload($files)
{
  for $name    in map:keys($files)
  let $content := $files($name)
  let $path    := file:temp-dir() || $name
  return (
    file:write-binary($path, $content),
    <file name="{ $name }" size="{ file:size($path) }"/>
  )
};
```

HTTP Headers

Header parameters are specified the same way as query parameters:

```
%rest:header-param("User-Agent", "{$user-agent}")
%rest:header-param("Referer", "{$referer}", "none")
```

Cookies

Cookie parameters are specified the same way as query parameters:

```
%rest:cookie-param("username", "{$user}")
%rest:cookie-param("authentication", "{$auth}", "no_auth")
```

Response

By default, a successful request is answered with the HTTP status code `200` (OK) and is followed by the given content. An erroneous request leads to an error code and an optional error message (e.g. `404` for “resource not found”).

Custom Response

Custom responses can be built from within XQuery by returning an `rest:response` element, an `http:response` child node that matches the syntax of the EXPath HTTP Client Module specification, and more optional child nodes that will be serialized as usual. A function that reacts on an unknown resource may look as follows:

```
declare %rest:path("") function page:error404() {
  <rest:response>
    <http:response status="404" message="I was not found.">
      <http:header name="Content-Language" value="en"/>
      <http:header name="Content-Type" value="text/html; charset=utf-8"/>
    </http:response>
  </rest:response>
};
```

Forwards and Redirects

The two XML elements `rest:forward` and `rest:redirect` can be used in the context of Web Applications, precisely in the context of RESTXQ. These nodes allow e.g. multiple XQuery Updates in a row by redirecting to the RESTXQ path of updating functions. Both wrap a URL to a RESTXQ path. The wrapped URL should be properly encoded via `fn:encode-for-uri()`.

Note that, currently, these elements are not part of RESTXQ specification.

rest:forward

Usage: wrap the location as follows

```
<rest:forward>{ $location }</rest:forward>
```

This results in a server-side forwarding, which as well reduces traffic among client and server. A forwarding of this kind will not change the URL seen from the client's perspective.

As an example, returning

```
<rest:forward>/hello/universe</rest:forward>
```

would internally forward to http://localhost:8984/hello/universe

rest:redirect

The function `web:redirect` can be used to create a redirect response element. Alternatively, the following element can be sent:

```
<rest:redirect>{ $location }</rest:redirect>
```

It is an abbreviation for:

```
<rest:response>
  <http:response status="302">
    <http:header name="location" value="{ $location }"/>
  </http:response>
</rest:response>
```

The client decides whether to follow this redirection. Browsers usually will, tools like curl won't unless `-L` is specified.

Output

The content-type of a response can be influenced by the user via Serialization Parameters. The steps are described in the REST chapter. In RESTXQ, serialization parameters can be specified in the query prolog, via annotations, or within the REST response element:

Query Prolog

In main modules, serialization parameters may be specified in the query prolog. These parameters will then apply to all functions in a module. In the following example, the content type of the response is overwritten with the `media-type` parameter:

```
declare option output:media-type 'text/plain';

declare %rest:path("version1") function page:version1() {
  'Keep it simple, stupid'
};
```

Annotations

Global serialization parameters can be overwritten via `%output` annotations. The following example serializes XML nodes as JSON, using the JsonML format:

```
declare
  %rest:path("cities")
  %output:method("json")
  %output:json("format=jsonml")
  function page:cities()
{
  element cities {
    db:open('factbook')//city/name
  }
};
```

The next function, when called, generates XHTML headers, and `text/html` will be set as content type:

```
declare
  %rest:path("")
  %output:method("xhtml")
  %output:omit-xml-declaration("no")
  %output:doctype-public("-//W3C//DTD XHTML 1.0 Transitional//EN")
  %output:doctype-system("http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd")
  function page:html()
{
  <html xmlns="http://www.w3.org/1999/xhtml">
    <body>done</body>
  </html>
};
```

Response Element

Serialization parameters can also be specified in a REST reponse element in a query. Serialization parameters will be overwritten:

```
declare %rest:path("version3") function page:version3() {
  <rest:response>
```

```
    <output:serialization-parameters>
      <output:media-type value='text/plain'/>
    </output:serialization-parameters>
  </rest:response>,
  'Not that simple anymore'
};
```

Error Handling

XQuery Errors

XQuery runtime errors can be processed via *error annotations*. Error annotations have one or more arguments, which represent the error codes to be caught. The codes equal the names of the XQuery 3.0 try/catch construct:

Priority	Syntax	Example
1	`prefix:name` `Q{uri}name`	`err:FORG0001` `Q{http://www.w3.org/2005/xqt-errors}FORG0001`
2	`prefix:*` `Q{uri}*`	`err:*` `Q{http://www.w3.org/2005/xqt-errors}*`
3	`*:name`	`*:FORG0001`
4	`*`	`*`

All error codes that are specified for a function must be of the same priority. The following rules apply when catching errors:

- Codes with a higher priority will be preferred.
- A global RESTXQ error will be raised if two functions with conflicting codes are found.

Similar to try/catch, the pre-defined variables (`code`, `description`, `value`, `module`, `line-number`, `column-number`, `additional`) can be bound to variables via *error parameter annotations*, which are specified the same way as query parameters.

Errors may occur unexpectedly. However, they can also be triggered by a query, as demonstrated by the following example:

```
declare
  %rest:path("/check/{$user}")
  function page:check($user)
{
  if($user = ('jack', 'lisa'))
  then 'User exists'
  else fn:error(xs:QName('err:user'), $user)
};

declare
  %rest:error("err:user")
  %rest:error-param("description", "{$user}")
  function page:user-error($user)
{
  'User "' || $user || '" is unknown'
};
```

HTTP Errors

Errors that occur outside RESTXQ can be caught by adding `error-page` elements with an error code and a target location to the `web.xml` configuration file (find more details in the Jetty Documentation):

```
<error-page>
  <error-code>404</error-code>
  <location>/error404</location>
</error-page>
```

The target location may be another RESTXQ function. The request:attribute function can be used to request details on the caught error:

```
declare %rest:path("/error404") function page:error404() {
  "URL: " || request:attribute("javax.servlet.error.request_uri") || ", " ||
  "Error message: " || request:attribute("javax.servlet.error.message")
};
```

Functions

The Request Module contains functions for accessing data related to the current HTTP request. Two modules exist for setting and retrieving server-side session data of the current user (Session Module) and all users known to the HTTP server (Sessions Module). The RESTXQ Module provides functions for requesting RESTXQ base URIs and generating a WADL description of all services. Please note that the namespaces of all of these modules must be explicitly specified via module imports in the query prolog.

The following example returns the current host name:

```
import module namespace request = "http://exquery.org/ns/request";

declare %rest:path("/host-name") function page:host() {
  'Remote host name: ' || request:remote-hostname()
};
```

References

Currently, the following external resources on RESTXQ exist:

- RESTXQ Specification , First Draft
- RESTful XQuery, Standardised XQuery 3.0 Annotations for REST . Paper, XMLPrague, 2012
- RESTXQ . Slides, MarkLogic User Group London, 2012
- Web Application Development . Slides from XMLPrague 2013

Changelog

Version 8.1

- Added: support for input-specific content-type parameters
- Added: `%input` annotations

Version 8.0

- Added: Support for regular expresssions in the Path Annotation
- Added: Evaluation of quality factors that are supplied in the Accept header

Version 7.9

- Updated: XQuery Errors, extended error annotations
- Added: `%rest:method`

Version 7.7

- Added: Error Handling, File Uploads, Multipart Types
- Updated: RESTXQ function may now also be specified in main modules (suffix: `*.xq`).
- Updated: the RESTXQ prefix has been changed from `restxq` to `rest`.
- Updated: parameters are implicitly cast to the type of the function argument
- Updated: the RESTXQ root url has been changed to `http://localhost:8984/`

Version 7.5

- Added: new XML elements `<rest:redirect/>` and `<rest:forward/>`

Chapter 78. REST

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the REST API of BaseX.

BaseX offers a RESTful API for accessing database resources via URLs. REST (REpresentational State Transfer) 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.

Usage

By default, REST services are available at `http://localhost:8984/rest/`. If no default credentials are specified in the URL or when starting the web application, they will be requested by the client (see further).

A web browser can be used to perform simple GET-based REST requests and display the response. 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 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.

Parameters

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.

While **Options** can be specified for all operations, the remaining parameters will only make sense for **Query** and **Run**.

Request

GET Method

If the GET method is used, all query parameters are directly specified within the URL. Additionally, the following **operations** can be specified:

- `query` : Evaluate an XQuery expression. If a database or database path is specified in the URL, it is used as initial query context.
- `command` : Execute a single database command.
- `run` : Evaluate an XQuery file or command script located on the server. The file path is resolved against the directory specified by `RESTPATH` (before, it was resolved against `WEBPATH`).

Examples

- Lists all resources found in the **tmp** path of the *factbook* database:`http://localhost:8984/rest/factbook/tmp`
- Serializes a document as JSONML:`http://localhost:8984/rest/factbook/factbook.xml?method=json&json=format=jsonml`
- `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 lists all database users that are known to BaseX:`http://localhost:8984/rest?command=show+users`

POST Method

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 `WEBPATH`:

```
<run xmlns="http://basex.org/rest">
  <text>query.xq</text>
</run>
```

PUT Method

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** : Incoming data is converted to XML if a parser is available for the specified content-type. The following content types are supported:

  - `application/json` : Stores JSON 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.

Conversion can be influenced by specifying additional content-type parameters (see RESTXQ for more information).

If raw data is added and if no content type, or a wrong content, is specified, a `400` (BAD REQUEST) error will be raised.

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://localhost:8984/rest/XMark`

- A new database is created, and no whitespaces will be removed from the passed on XML input:`http://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://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 Method

The DELETE method is used to delete databases or resources within a database.

Example

- The **factbook** database is deleted:`http://localhost:8984/rest/factbook`
- All resources of the **XMark** database are deleted that reside in the **tmp** path:`http://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 Method

All query parameters that have not been processed before will be treated as variable assignments:

Example

- The following request assigns two variables to a server-side query file `mult.xq` placed in the HTTP directory:`http://localhost:8984/rest?run=mult.xq&$a=21&$b=2`

```
(: XQuery file: mult.xq :)
declare variable $a as xs:integer external;
declare variable $b as xs:integer external;
<mult>{ $a * $b }</mult>
```

The dollar sign can be omitted as long as the variable name does not equal a parameter keyword (e.g.: `method`).

POST Method

If `query` or `run` is used as operation, external variables can be specified via the `<variable/>` element:

```
<query xmlns="http://basex.org/rest">
  <text><![CDATA[
    declare variable $x as xs:integer external;
    declare variable $y as xs:integer external;
    <mult>{ $a * $b }</mult>
  ]]></text>
  <variable name="a" value="21"/>
  <variable name="b" value="2"/>
</query>
```

Response

Content Type

As the content type of a REST response cannot necessarily be dynamically determined, it can be enforced by the user. The final content type of a REST response is chosen as follows:

1. If the serialization parameter `media-type` is supplied, it will be adopted as content-type.

2. Otherwise, if the serialization parameter `method` is supplied, the content-type will be chosen according to the following mapping:
   - `xml` → `application/xml`
   - `xhtml` → `text/html`
   - `html` → `text/html`
   - `text` → `text/plain`
   - `json` → `application/json`
   - `raw` → `application/octet-stream` (binary data will be sent in its original byte representation, i.#e., without further conversion)
3. If no media-type or serialization method is supplied, the content type of a response depends on the chosen REST operation:
   - **Query** /**Run** → `application/xml`
   - **Command** → `text/plain`
   - **Get** → `application/xml`, or content type of the addressed resource

Serialization parameters can either be supplied as query parameters or within the query.

The following three example requests all return `<a/>` with `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:

```
import java.net.*;
import org.basex.util.*;

public final class RESTExample {
  public static void main(String[] args) throws Exception {
    // The java URL connection to the resource.
    URL url = new URL("http://localhost:8984/rest/factbook");

    // Establish the connection to the URL.
    HttpURLConnection conn = (HttpURLConnection) 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, POST, PUT, DELETE.

Command Line

Tools such as the Linux commands Wget or cURL exist to perform HTTP requests (try copy & paste):

GET

- `curl -i "http://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>""http://localhost:8984/rest/factbook"`
- `curl -i -X POST -H "Content-Type: application/xml" -T query.xml "http://localhost:8984/rest/factbook"`

PUT

- `curl -i -X PUT -T "etc/xml/factbook.xml""http://localhost:8984/rest/factbook"`
- `curl -i -X PUT -H "Content-Type: application/json" -T "plain.json""http://localhost:8984/rest/plain"`

DELETE

- `curl -i -X DELETE "http://admin:admin@localhost:8984/rest/factbook"`

Changelog

Version 8.1

- Added: support for input-specific content-type parameters

- Updated: the run operation now resolves file paths against the RESTPATH option.

Version 8.0

- Removed: `wrap` parameter

Version 7.9

- Updated: Also evaluate command scripts via the `run` operation.

Version 7.2

- Removed: direct evaluation of adresses 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

Chapter 79. REST: POST Schema

Read this entry online in the BaseX Wiki.

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>
```

Chapter 80. WebDAV

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the WebDAV file system interface.

BaseX offers access to the databases and documents using the WebDAV 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.

Usage

By default, the BaseX HTTP server makes the WebDAV service accessible at `http://localhost:8984/webdav/`. If no default credentials are specified, they will be requested by the client (see further). It can be accessed by either `http://<httphost>:<httpport>/webdav/` or `webdav://<httphost>:<httpport>/webdav/`, depending on your WebDAV client.

Please note that the file size of XML documents will be displayed as 0 bytes, as the actual file size can only be determined if the full document is being returned and serialized. This may cause problems with some WebDAV clients (e.g. NetDrive or WebDrive).

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.

Root Directory

In the WebDAV root directory, all existing databases are listed. As new resources can only be stored within database, it is not possible to store files in the root directory. If a file is copied on top level, a new database will be created, which contains this resource.

Locking

The BaseX WebDAV implementation supports locking. It can be utilized with clients which support this feature (e.g. oXygen Editor). EXCLUSIVE and SHARED locks are supported, as well as WRITE locks.

Note: WebDAV locks are stored in a database called `~webdav`. If the database is deleted, it will automatically be recreated along with the next lock operations. If a resource remains locked, it can be unlocking by removing the correspondent `<w:lockinfo>` entry.

WebDAV Clients

Please check out the following tutorials to get WebDAV running on different operating systems and with oXygen:

- Windows 7 and up
- Windows XP
- Mac OSX 10.4+
- GNOME and Nautilus
- KDE
- oXygen Editor

Changelog

Version 7.7

- Added: Locking

Version 7.0

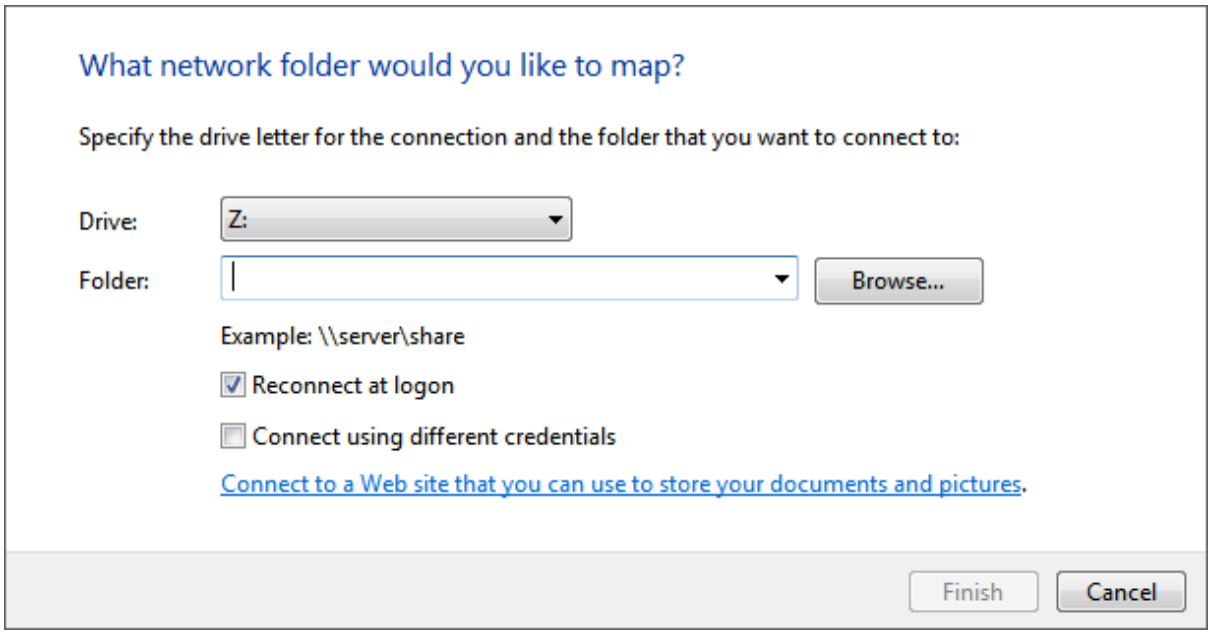
- WebDAV API introduced

Chapter 81. WebDAV: Windows 7

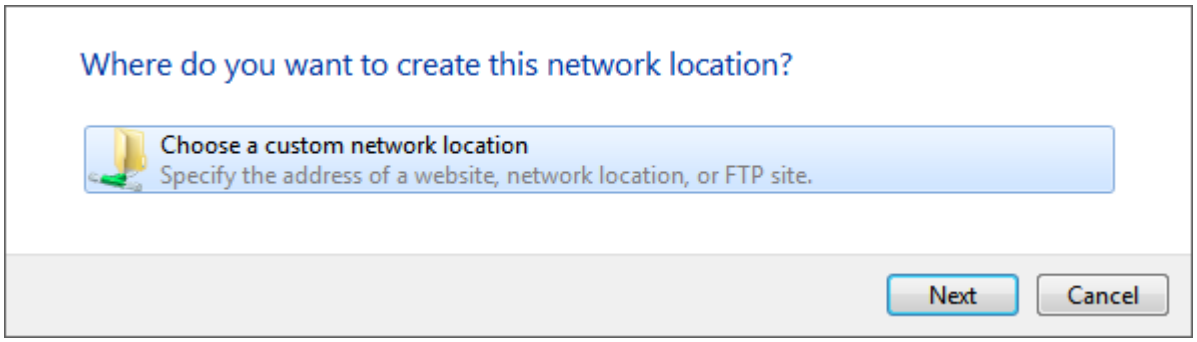
Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows 7.

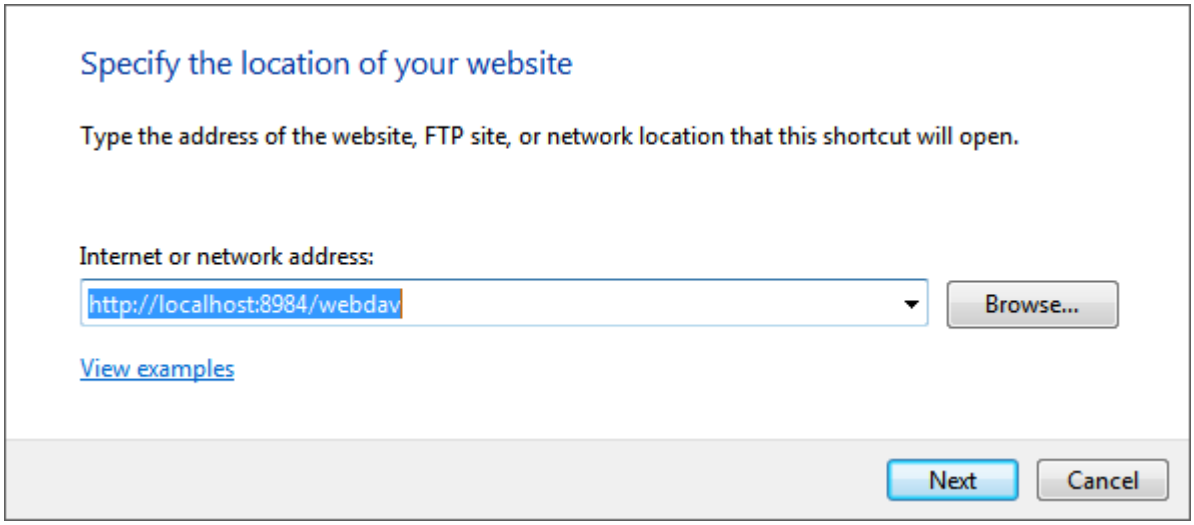
- Open the Explorer
- Open the "Map network drive..." dialog by right-clicking on "My Computer"
- Click on the link "Connect to a Web site that you can use to store your documents and pictures."



- Click "Next", select "Choose a custom network location" and click "Next" again.

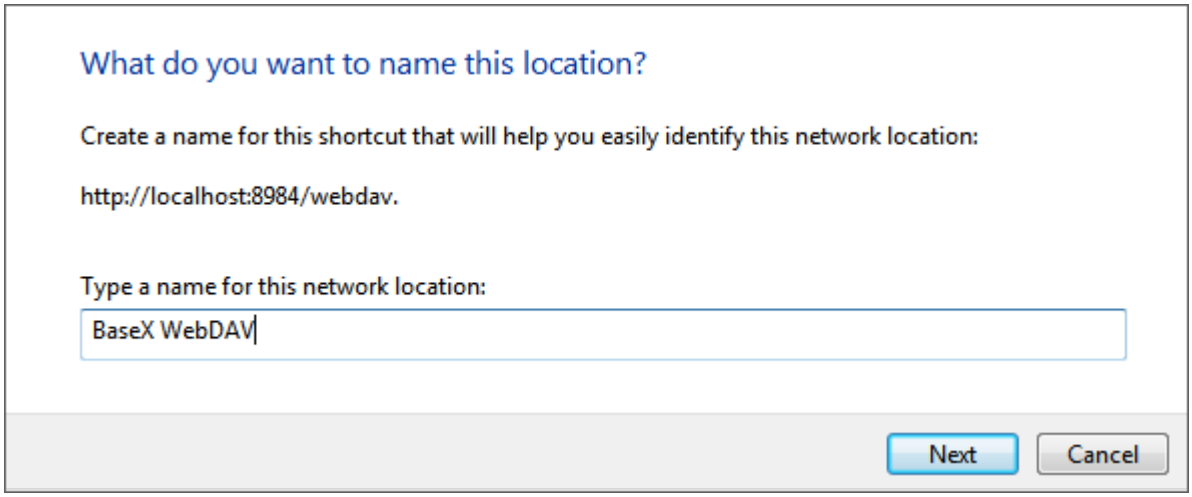


- Enter the URL address of the BaseX WebDAV Server (e.g. `http://localhost:8984/webdav`) and click "Next".

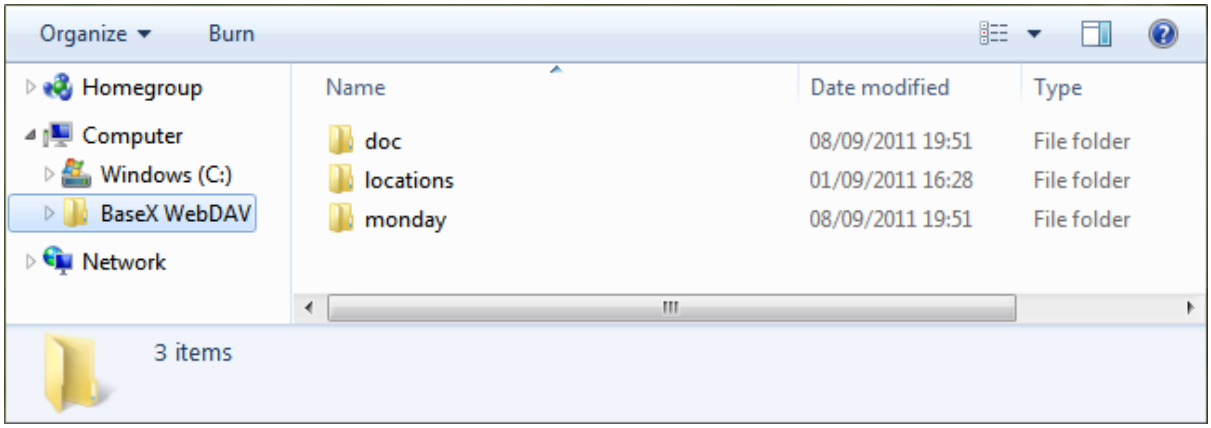


If a message saying that the folder is not valid, this is because Microsoft WebClient is not configured to use Baseic HTTP authentication. Please check this Microsoft article in order to enable Basic HTTP authentication.

- Enter a name for the network location and click "Next".



- The BaseX WebDAV can be accessed from the Explorer window.

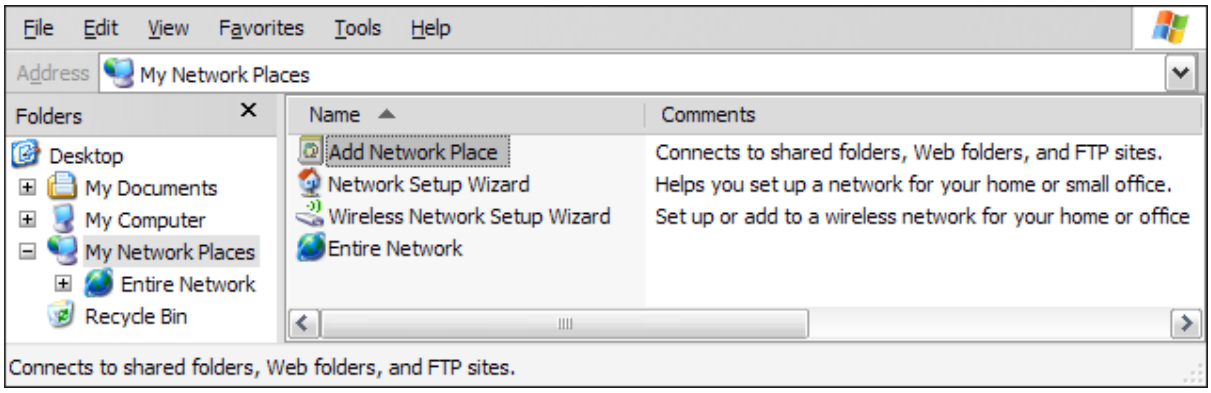


Chapter 82. WebDAV: Windows XP

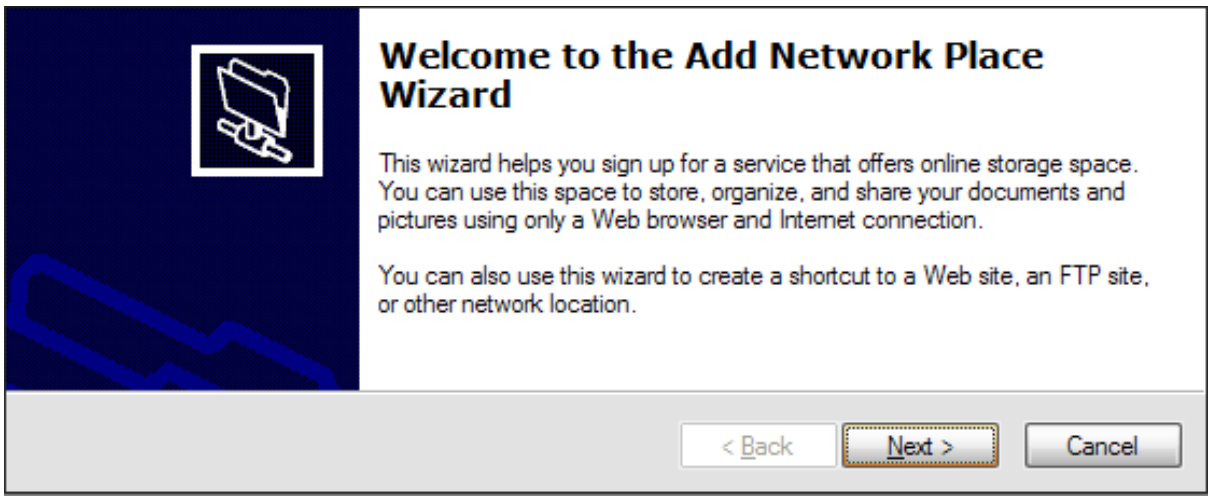
Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows XP.

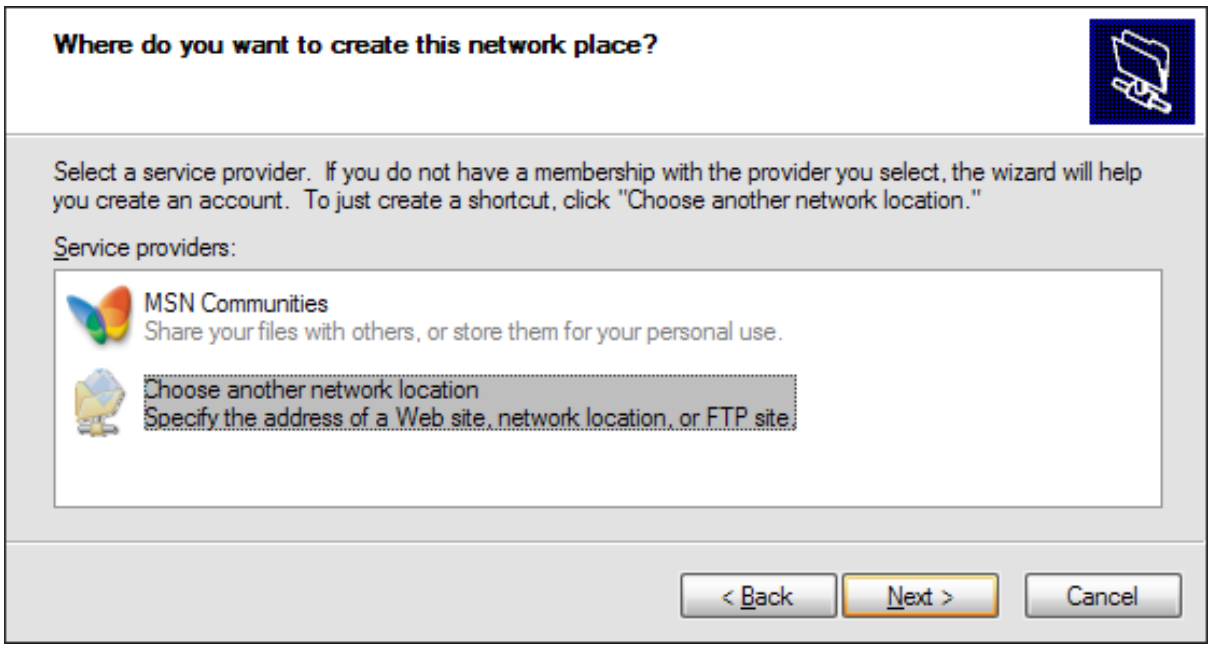
- In the "My Network Places" view, double click on "Add Network Place":



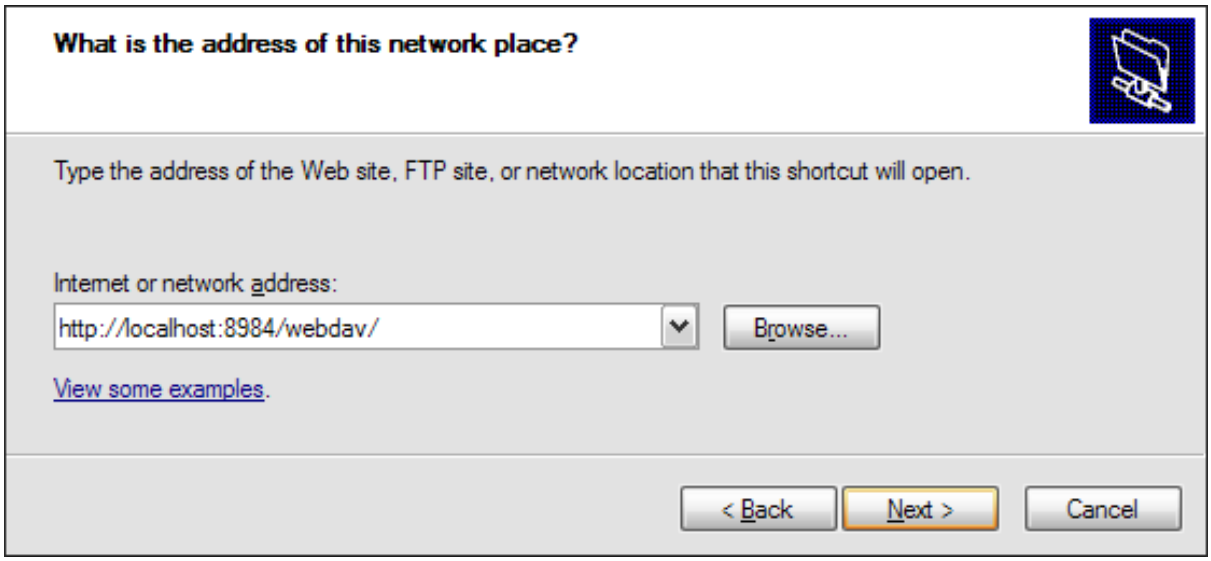
- Confirm the upcoming introductory dialog:



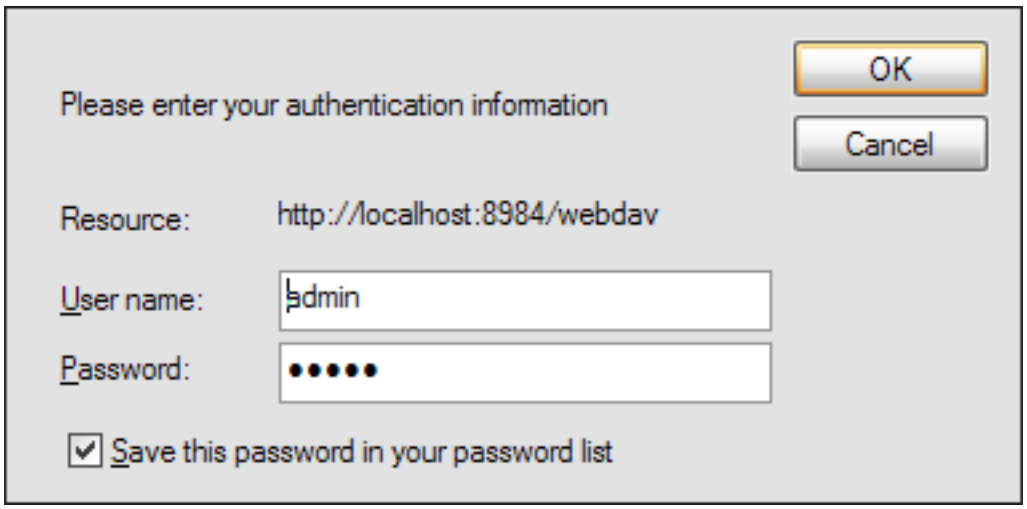
- Select "Choose another network location" in the next dialog:



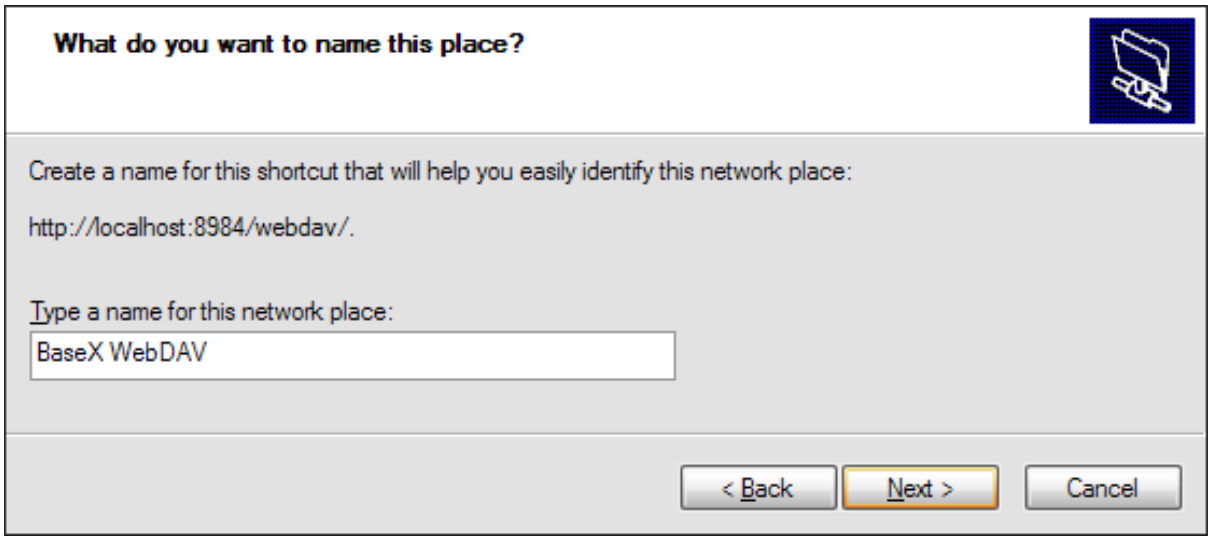
- Next, specify the BaseX WebDAV URL:



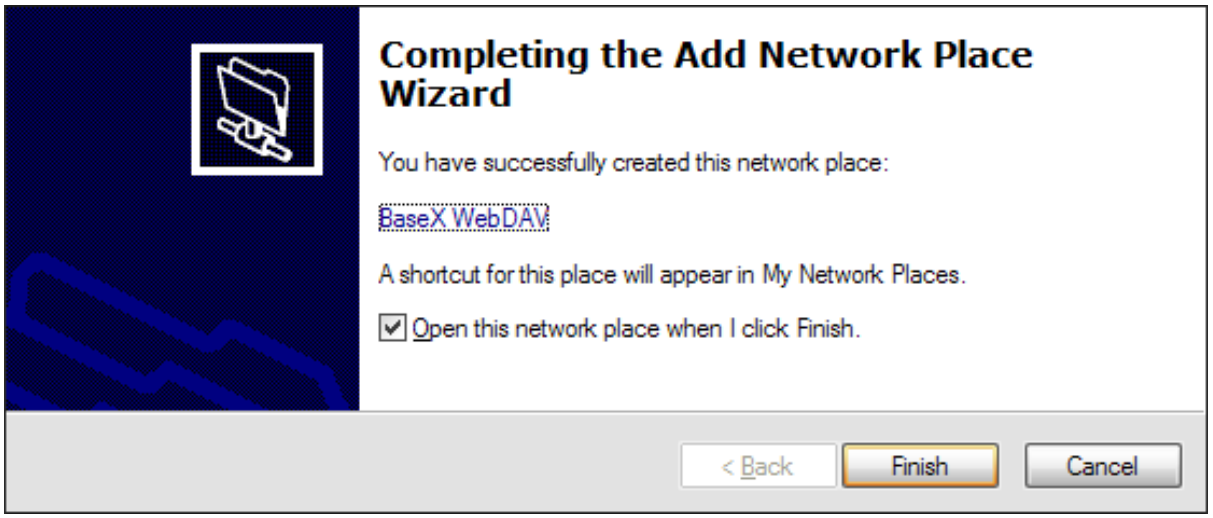
- Enter the user/password combination to connect to the WebDAV service:



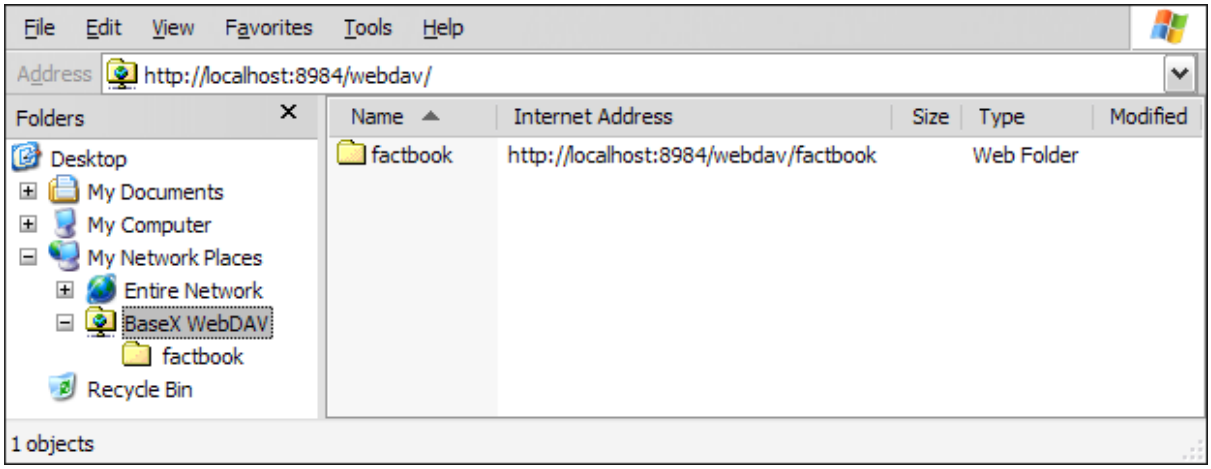
- Assign a name to your WebDAV connection:



- Finish the wizard:



- You can now see all BaseX databases in the Windows Explorer:

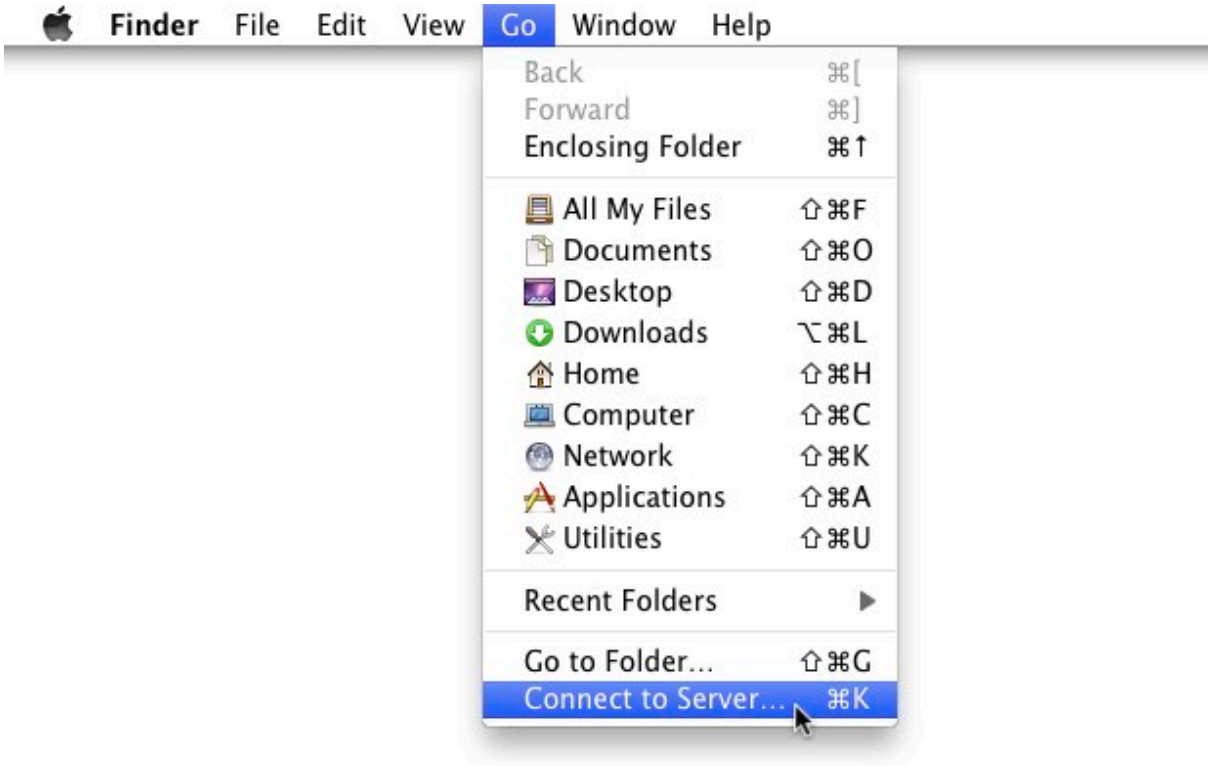

File Edit View Favorites Tools Help
Address http://localhost:8984/webdav/
Folders
Desktop
My Documents
My Computer
My Network Places
Entire Network
BaseX WebDAV
factbook
Recycle Bin
Name
Internet Address
Size
Type
Modified
factbook
http://localhost:8984/webdav/factbook
Web Folder
1 objects

Chapter 83. WebDAV: Mac OSX

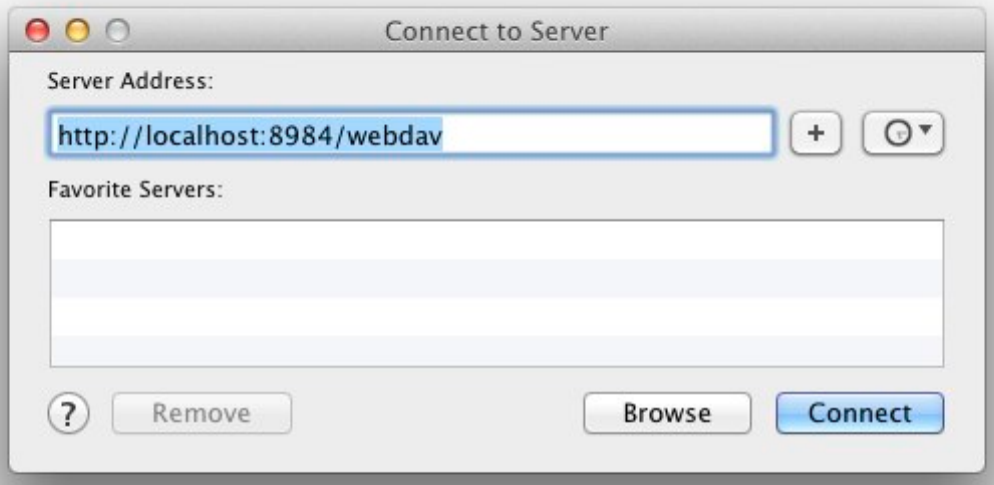
Read this entry online in the BaseX Wiki.

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:

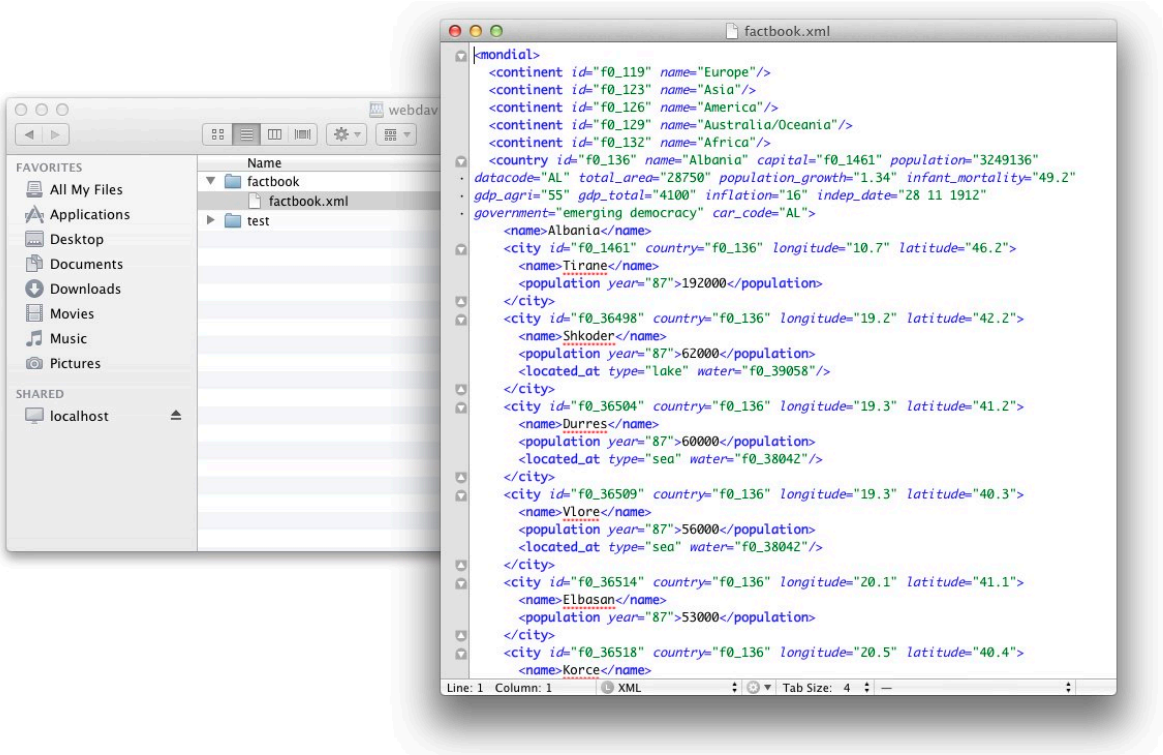


- Enter BaseX WebDAV URL (eg. http://localhost:8984/webdav) - do not use webdav://-scheme! Press Connect:



- Enter the user credentials:

- That's it, now the databases can be browsed:

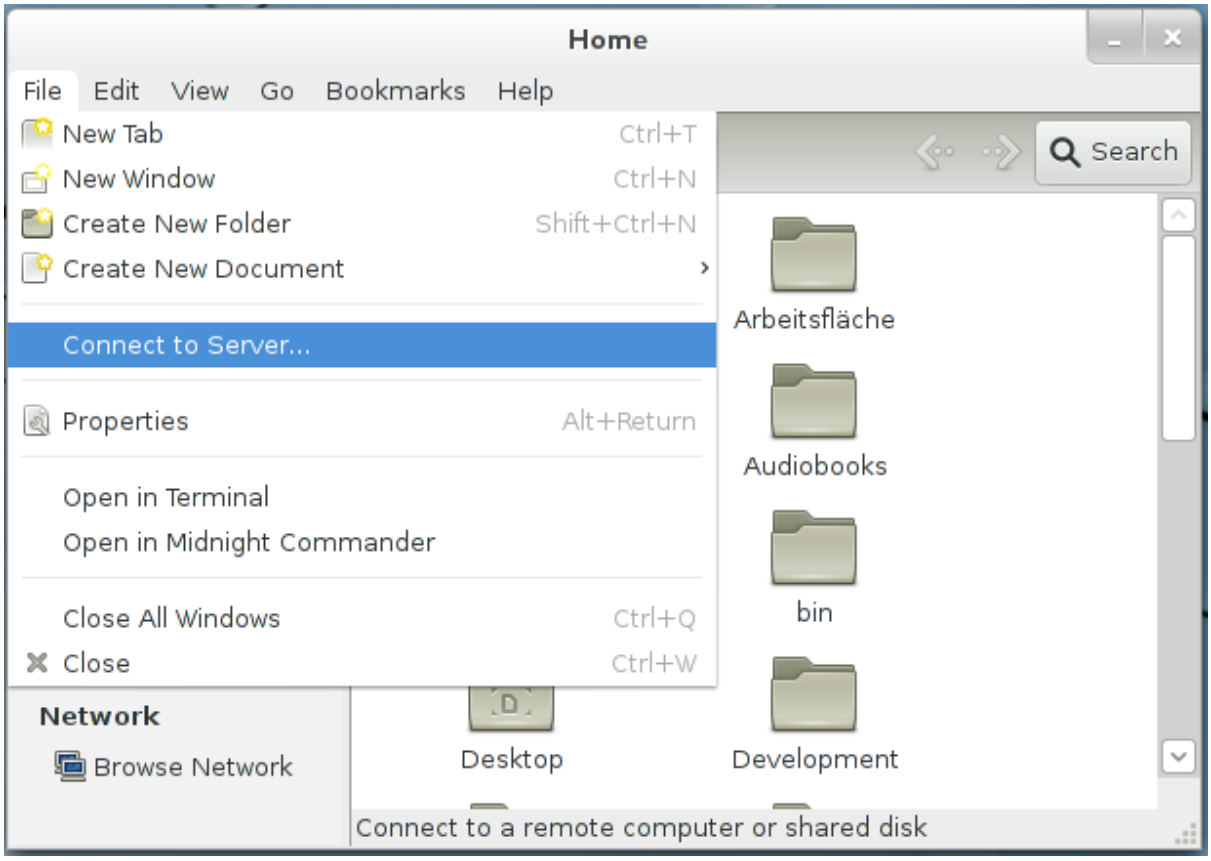

webdav
FAVORITES
All My Files
Applications
Desktop
Documents
Downloads
Movies
Music
Pictures
SHARED
localhost
Name
factbook
factbook.xml
test
factbook.xml
<mondial>
<continent id="f0_119" name="Europe"/>
<continent id="f0_123" name="Asia"/>
<continent id="f0_126" name="America"/>
<continent id="f0_129" name="Australia/Oceania"/>
<continent id="f0_132" name="Africa"/>
<country id="f0_136" name="Albania" capital="f0_1461" population="3249136"
datacode="AL" total_area="28750" population_growth="1.34" infant_mortality="49.2"
gdp_agri="55" gdp_total="4100" inflation="16" indep_date="28 11 1912"
government="emerging democracy" car_code="AL">
<name>Albania</name>
<city id="f0_1461" country="f0_136" longitude="10.7" latitude="46.2">
<name>Tirane</name>
<population year="87">192000</population>
</city>
<city id="f0_36498" country="f0_136" longitude="19.2" latitude="42.2">
<name>Shkoder</name>
<population year="87">62000</population>
<located_at type="lake" water="f0_39058"/>
</city>
<city id="f0_36504" country="f0_136" longitude="19.3" latitude="41.2">
<name>Durres</name>
<population year="87">60000</population>
<located_at type="sea" water="f0_38042"/>
</city>
<city id="f0_36509" country="f0_136" longitude="19.3" latitude="40.3">
<name>Vlore</name>
<population year="87">56000</population>
<located_at type="sea" water="f0_38042"/>
</city>
<city id="f0_36514" country="f0_136" longitude="20.1" latitude="41.1">
<name>Elbasan</name>
<population year="87">53000</population>
</city>
<city id="f0_36518" country="f0_136" longitude="20.5" latitude="40.4">
<name>Korce</name>
Line: 1 Column: 1
XML
Tab Size: 4

Chapter 84. WebDAV: GNOME

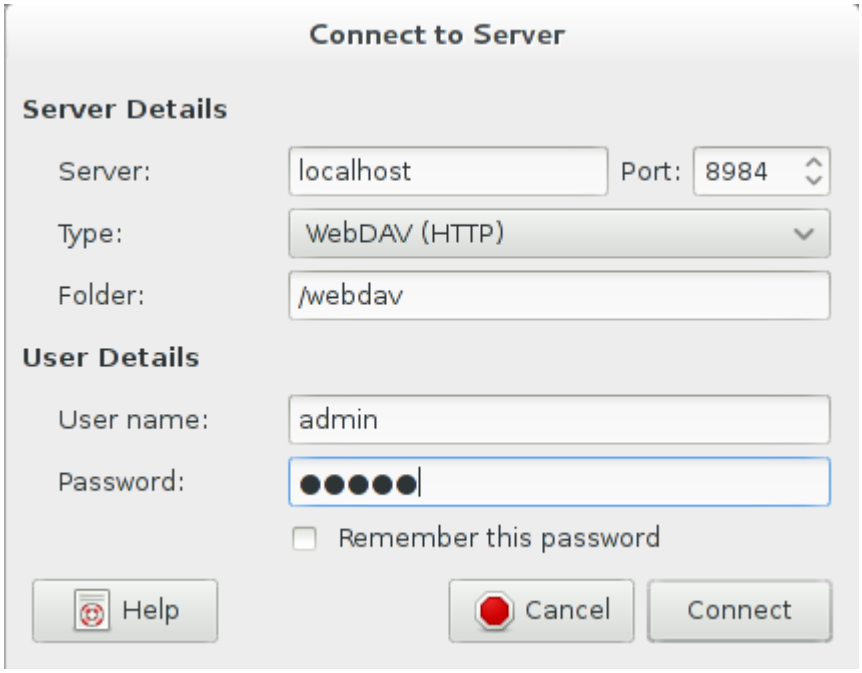
Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with GNOME and Nautilus.

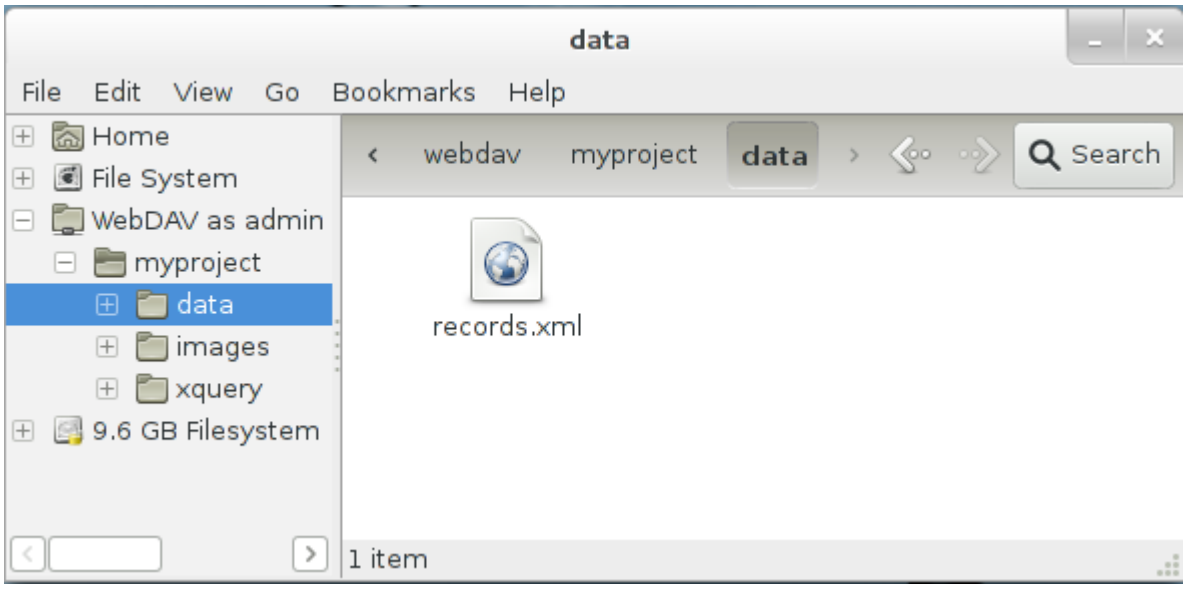
- In Nautilus choose File -> Connect to Server:



- 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:

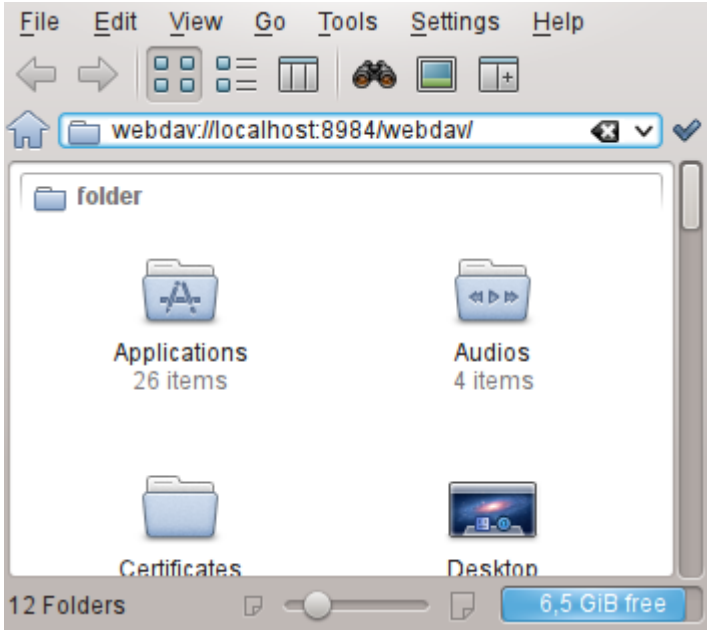


Chapter 85. WebDAV: KDE

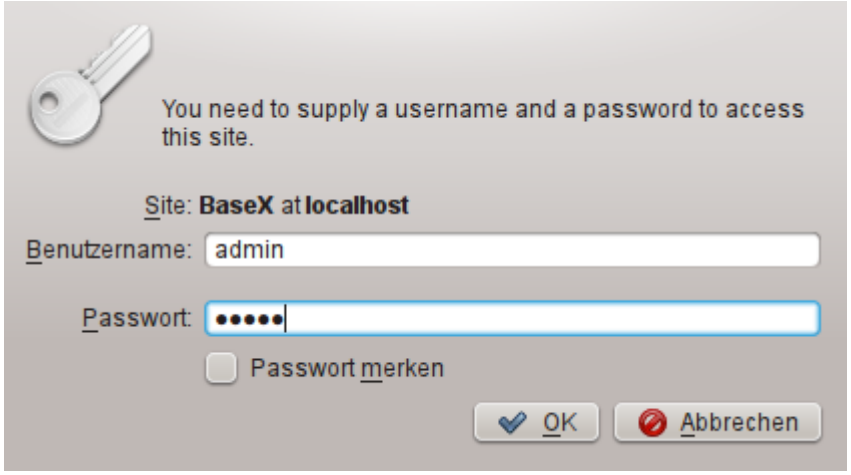
Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with KDE.

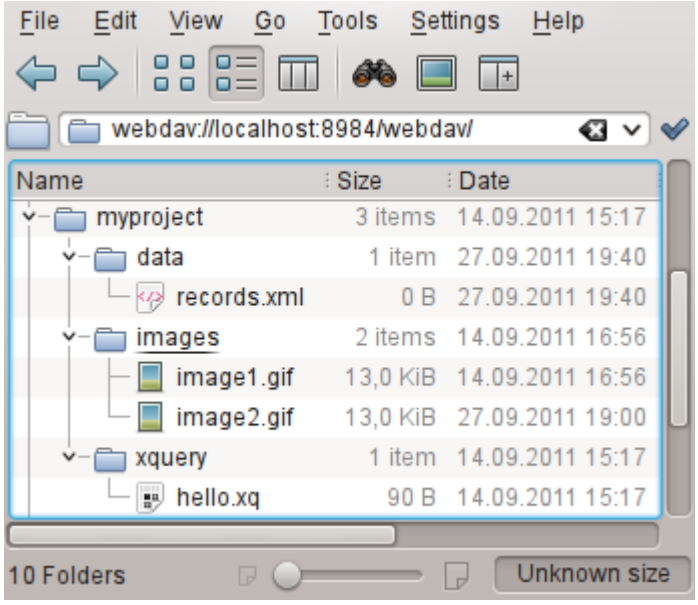
- KDE SC provides two file managers - Dolphin and Konqueror, which both support WebDAV using the "webdav://" URL prefix. Start Dolphin or Konqueror and enter the BaseX WebDAV URL (eg. webdav://localhost:8984/webdav):



- Enter the user credentials:



- After clicking "OK" the databases can be browsed:


File
Edit
View
Go
Tools
Settings
Help
webdav://localhost:8984/webdav/
Name
Size
Date
myproject
3 items
14.09.2011 15:17
data
1 item
27.09.2011 19:40
records.xml
0 B
27.09.2011 19:40
images
2 items
14.09.2011 16:56
image1.gif
13,0 KiB
14.09.2011 16:56
image2.gif
13,0 KiB
27.09.2011 19:00
xquery
1 item
14.09.2011 15:17
hello.xq
90 B
14.09.2011 15:17
10 Folders
Unknown size

Part IX. Client APIs

Chapter 86. Clients

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to communicate with BaseX from other programming languages.

You can use the following light-weight language bindings to connect to a running BaseX server instance, execute database commands and evaluate XQuery expressions.

Most clients provide two modes:

- Standard Mode : connecting to a server, sending commands
- Query Mode : defining queries, binding variables, iterative evaluation

Currently, we offer bindings for the following programming languages:

BaseX 7.x, BaseX 8.x and later

- Java : The default implementation
- C++ : contributed by Jean-Marc Mercier
- node.js : contributed by Andy Bunce
- Perl , contributed by the BaseX Team
- PHP : updated by James Ball
- Python 3.x, 2.7.3 : contributed by Hiroaki Itoh
- Python , using BaseX REST services: contributed by Luca Lianas
- Ruby , contributed by the BaseX Team

With **Version 8.0**, authentication has changed. Some of the language bindings have not been updated yet. The update is rather trivial, though (see here for more details); we are looking forward to your patches!

BaseX 7.x (outdated)

- ActionScript : contributed by Manfred Knobloch
- C , contributed by the BaseX Team
- C# , contributed by the BaseX Team
- Golang : contributed by Christian Baune
- Haskell : contributed by Leo Wörteler
- Lisp : contributed by Andy Chambers
- node.js : contributed by Hans Hübner (deviating from client API)
- Python < 2.7 : improved by Arjen van Elteren
- Qt : contributed by Hendrik Strobelt
- Rebol : contributed by Sabu Francis
- Scala : contributed by Manuel Bernhardt
- Scala (simple implementation)
- VB , contributed by the BaseX Team

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.

Changelog

Version 8.0

- Updated: cram-md5 replaced with digest authentication

Chapter 87. Standard Mode

Read this entry online in the BaseX Wiki.

In the standard mode of the Clients, a database command can be sent to the server using the `execute()` function of the `Session`. This functions 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:

```
<?php
/*
 * This example shows how database commands can be executed.
 * Documentation: http://basex.org/api
 *
 * (C) BaseX Team 2005-15, 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();
}
?>
```

Chapter 88. Query Mode

Read this entry online in the BaseX Wiki.

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. 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:

```
<?php
/*
 * This example shows how queries can be executed in an iterative manner.
 * Documentation: http://basex.org/api
 *
 * (C) BaseX Team 2005-15, 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();
}
?>
```

Changelog

Version 7.2

- Added: `context()` function

Chapter 89. Server Protocol

Read this entry online in the BaseX Wiki.

This page presents the classes and functions of the BaseX Clients, and the underlying protocol, which is utilized for communicating with the database server. A detailed example demonstrates how a concrete byte exchange can look like.

Workflow

- All clients are based on the client/server architecture. Hence, a BaseX database server must be started first.
- 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 and the context item can be bound to that instance, 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.

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 your own client.

Conventions

- `\x` : single byte.
- `{...}` : utf8 strings or raw data, suffixed with a `\0` byte. To avoid confusion with this end-of-string byte, all `\0` and `\FF` bytes that occur in raw data will be prefixed with `\FF`.

Authentication

Digest

Digest authentication is used since Version 8.0:

1. Client connects to server socket
2. Server sends a **realm** and **nonce**, separated by a colon: `{realm:nonce}`
3. Client sends the **user name** and a hash value. The hash is composed of the md5 hash of
   a. the md5 hash of the **user name**, **realm**, and **password** (all separated by a colon), and
   b. the **nonce**: `{username} {md5(md5(username:realm:password) + nonce)}`
4. Server replies with `\0` (success) or `\1` (error)

CRAM-MD5

CRAM-MD5 was discarded, because unsalted md5 hashes could easily be uncovered using rainbow tables. However, most client bindings still provide support for the outdated handshaking, as it only slightly differs from the new protocol:

1. Client connects to server socket
2. Server sends a **nonce** (timestamp): `{nonce}`
3. Client sends the **user name** and a hash value. The hash is composed of the md5 hash of
   a. the md5 of the **password** and
   b. the **nonce**: `{username} {md5(md5(password) + nonce)}`
4. Server replies with `\0` (success) or `\1` (error)

Clients can easily be implemented to both support `digest` and `cram-md5` authentication: If the first server response contains no colon, `cram-md5` should be chosen.

Command Protocol

The following byte sequences are sent and received from the client (please note that a specific client may 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.
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		`{` *partial 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. The type will be ignored if the string is empty.
RESULTS	`\4 {id}`	`\x {item} ... \x {item} \0`	Returns all resulting items as strings, prefixed by a single byte (`\x`) that

			represents the Type ID. This command is called by the `more()` function of a client implementation.
EXECUTE	`\5 {id}`	`{result} \0`	Executes the query and returns the result as a single string.
INFO	`\6 {id}`	`{result} \0`	Returns a string with query compilation and profiling info.
OPTIONS	`\7 {id}`	`{result} \0`	Returns a string with all query serialization parameters, which can e.g. be assigned to the SERIALIZER option.
CONTEXT	`\14 {id} {value} {type}`	`\0 \0`	Binds a value to the context. The type will be ignored if the string is empty.
UPDATING	`\30 {id}`	`{result} \0`	Returns `true` if the query contains updating expressions; `false` otherwise.
FULL	`\31 {id}`	`XDM {item} ... XDM {item} \0`	Returns all resulting items as strings, prefixed by the XDM Meta Data. This command is e. g. used by the XQJ API.

As can be seen in the table, all results end with a single `\0` byte, which indicates that the process was successful. If an error occurs, an additional byte `\1` is sent, which is then followed by the `error` message string.

Binding Sequences

Also sequences can be bound to variables and the context:

- `empty-sequence()` must be supplied as type if an empty sequence is to be bound.
- Multiple items are supplied via the `{value}` argument and separated with `\1` bytes.
- Item types are specified by appending `\2` and the type in its string representation to an item. If no item type is specified, the general type is used.

Some examples for the `{value}` argument:

- the two integers `123` and `789` are encoded as `123`, `\1`, `789` and `\0` (`xs:integer` may be specified via the `{type}` argument).
- the two items `xs:integer(123)` and `xs:string('ABC')` are encoded as `123`, `\2`, `xs:integer`, `\1`, `ABC`, `\2`, `xs:string` and `\0`.

Example

In the following example, a client registers a new session and executes the INFO database command. Next, it creates a new query instance for the XQuery expression `1, 2+'3'`. The query is then evaluated, and the server returns the result of the first subexpression `1` and an error for the second sub expression. Finally, the query instance and client session are closed.

- **Client** connects to the database server socket
- **Server** sends realm and timestamp "BaseX:1369578179679": `# 42 61 73 65 58 3A 31 33 36 39 35 37 38 31 37 39 36 37 39 00`
- **Client** sends user name "jack": `6A 61 63 6B 00 #`
- **Client** sends hash: md5(md5("jack:BaseX:topsecret") + "1369578179679") = "ca664a31f8deda9b71ea3e79347f6666": `63 61 36 ... 00 #`
- **Server** replies with success code: `# 00`
- **Client** sends the "INFO" command: `49 4E 46 4F 00 #`
- **Server** responds with the result "General Information...": `# 47 65 6e 65 ... 00`
- **Server** additionally sends an (empty) info string: `# 00`
- **Client** creates a new query instance for the XQuery "1, 2+'3'": `00 31 2C 20 32 2B 27 33 27 00 #`
- **Server** returns query id "1" and a success code: `# 31 00 00`
- **Client** requests the query results via the RESULTS protocol command and its query id: `04 31 00 #`
- **Server** returns the first result ("1", type xs:integer): `# 52 31 00`
- **Server** sends a single "\0" byte instead of a new result, which indicates that no more results can be expected: `# 00`
- **Server** sends the error code "\1" and the error message ("Stopped at..."): `# 01 53 74 6f ... 00`
- **Client** closes the query instance: `02 31 00 #`
- **Server** sends a response (which is equal to an empty info string) and success code: `# 00 00`
- **Client** closes the socket connection

Constructors and Functions

Most language bindings provide the following constructors and functions:

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 instance for the specified query:`Query query(String query)`
- Create a database from an input stream:`void create(String name, InputStream input)`
- Add a document to the current database from an input stream:`void add(String path, InputStream input)`
- Replace a document with the specified input stream:`void replace(String path, InputStream input)`
- Store raw data at the specified path:`void store(String path, InputStream input)`
- Return process information:`String info()`
- Close the session:`void close()`

Query

- Create query instance with session and query:`Query(Session session, String query)`
- Bind an external variable:`void bind(String name, String value, String type)`The type can be an empty string.
- 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()`

Changelog

Version 8.2

- Removed: `WATCH` and `UNWATCH` command

Version 8.0

- Updated: cram-md5 replaced with digest authentication
- Updated: `BIND` command: support more than one item

Version 7.2

- Added: Query Commands CONTEXT, UPDATING and FULL
- Added: Client function `context(String value, String type)`

Chapter 90. Server Protocol: Types

Read this entry online in the BaseX Wiki.

This article lists extended type information that is returned by the Server Protocol.

XDM Meta Data

In most cases, the XDM meta data is nothing else than the Type ID. There are three exceptions, though: document-node(), attribute() and xs:QName items are followed by an additional {URI} string.

Type IDs

The following table lists the type IDs that are returned by the server. Currently, all node kinds are of type `xs:untypedAtomic`:

Type ID	Node Kind/Item Type	Type
7	Function item	*function*
8	`node()`	*node*
9	`text()`	*node*
10	`processing-instruction()`	*node*
11	`element()`	*node*
12	`document-node()`	*node*
13	`document-node(element())`	*node*
14	`attribute()`	*node*
15	`comment()`	*node*
32	`item()`	*atomic value*
33	`xs:untyped`	*atomic value*
34	`xs:anyType`	*atomic value*
35	`xs:anySimpleType`	*atomic value*
36	`xs:anyAtomicType`	*atomic value*
37	`xs:untypedAtomic`	*atomic value*
38	`xs:string`	*atomic value*
39	`xs:normalizedString`	*atomic value*
40	`xs:token`	*atomic value*
41	`xs:language`	*atomic value*
42	`xs:NMTOKEN`	*atomic value*
43	`xs:Name`	*atomic value*
44	`xs:NCName`	*atomic value*
45	`xs:ID`	*atomic value*
46	`xs:IDREF`	*atomic value*
47	`xs:ENTITY`	*atomic value*
48	`xs:float`	*atomic value*
49	`xs:double`	*atomic value*
50	`xs:decimal`	*atomic value*
51	`xs:precisionDecimal`	*atomic value*

52	`xs:integer`	*atomic value*
53	`xs:nonPositiveInteger`	*atomic value*
54	`xs:negativeInteger`	*atomic value*
55	`xs:long`	*atomic value*
56	`xs:int`	*atomic value*
57	`xs:short`	*atomic value*
58	`xs:byte`	*atomic value*
59	`xs:nonNegativeInteger`	*atomic value*
60	`xs:unsignedLong`	*atomic value*
61	`xs:unsignedInt`	*atomic value*
62	`xs:unsignedShort`	*atomic value*
63	`xs:unsignedByte`	*atomic value*
64	`xs:positiveInteger`	*atomic value*
65	`xs:duration`	*atomic value*
66	`xs:yearMonthDuration`	*atomic value*
67	`xs:dayTimeDuration`	*atomic value*
68	`xs:dateTime`	*atomic value*
69	`xs:dateTimeStamp`	*atomic value*
70	`xs:date`	*atomic value*
71	`xs:time`	*atomic value*
72	`xs:gYearMonth`	*atomic value*
73	`xs:gYear`	*atomic value*
74	`xs:gMonthDay`	*atomic value*
75	`xs:gDay`	*atomic value*
76	`xs:gMonth`	*atomic value*
77	`xs:boolean`	*atomic value*
78	`basex:binary`	*atomic value*
79	`xs:base64Binary`	*atomic value*
80	`xs:hexBinary`	*atomic value*
81	`xs:anyURI`	*atomic value*
82	`xs:QName`	*atomic value*
83	`xs:NOTATION`	*atomic value*

Chapter 91. Java Examples

Read this entry online in the BaseX Wiki.

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. by integrating the BaseX code. Most examples are taken from our basex-examples repository, in which you will find some more use cases.

Local Examples

The following code snippets work in *embedded* mode; they do not rely on an additional server instance:

- RunCommands.java creates and drops database and index instances, prints a list of all existing databases.
- RunQueries.java shows three variants of running queries.
- BindContext.java demonstrates how a value can be bound as context item.
- BindVariables.java demonstrates how a value can be bound to a variable.
- CreateCollection.java creates and manages a collection.
- QueryCollection.java creates, runs queries against it and drops a collection.
- WikiExample.java creates a database from an url (wiki instance), runs a query against it and drops the database.

Server Examples

The examples below take advantage of the client/server architecture:

- ServerCommands.java launches server-side commands using a client session.
- ServerAndLocal.java processes server results locally.
- ServerConcurrency.java runs concurrent queries.
- ServerQueries.java shows how iterative queries can be performed.
- UserExample.java manages database users.

XQuery Module Examples

BaseX provides Java Bindings for accessing external Java code via XQuery functions. The following examples show how this feature can be utilized:

- FruitsExample.java demonstrates how Java classes can be imported as XQuery modules.
- FruitsModule.java is a simple demo module called by `FruitsExample`.
- ModuleDemo.java is a simple XQuery demo module that demonstrates how XQuery items can be processed from Java. It is derived from the `QueryModule` class.
- QueryModule.java is located in the BaseX core. Java query modules can extend this class to get access to the current query context and enrich functions with properties ().

XQJ API

The implementation of the BaseX XQJ API (closed-source) has been written by Charles Foster. It uses the client/server architecture. The basex-examples repository contains various examples on how to use XQJ.

Client API

- BaseXClient.java provides an implementation of the Server Protocol.
- Example.java demonstrates how commands can be executed on a server.
- QueryExample.java shows how queries can be executed in an iterative manner.
- QueryBindExample.java shows how external variables can be bound to XQuery expressions.
- CreateExample.java shows how new databases can be created.
- AddExample.java shows how documents can be added to databases, and how existing documents can be replaced.
- BinaryExample.java shows how binary resource can be added to and retrieved from the database.

REST API

- RESTGet.java presents the HTTP GET method.
- RESTPost.java presents the HTTP POST method.
- RESTPut.java presents the HTTP PUT method.
- RESTAll.java runs all examples at one go.

XML:DB API (deprecated)

Note that the XML:DB API does not talk to the server and can thus only be used in embedded mode.

- XMLDBCreate.java creates a collection using XML:DB.
- XMLDBQuery.java runs a query using XML:DB.
- XMLDBInsert.java inserts a document into a database using XML:DB.

Part X. Advanced User's Guide

Chapter 92. Advanced User's Guide

Read this entry online in the BaseX Wiki.

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
- Storage Layout : How data is stored in the database files

Use Cases

- Statistics : Exemplary statistics on databases created with BaseX
- Twitter : Storing live tweets in BaseX

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
- Execution Plan : Analyzing query evaluation

Chapter 93. Configuration

Read this entry online in the BaseX Wiki.

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.
- `.basexhistory` contains commands that have been typed in most recently.
- `.basexhome` can be created by a user to mark a folder as home directory. Its contents do not matter, so it is usually empty.

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` or `.basexhome` file is found in this directory.
- Otherwise, the files are searched in the **application directory** (the folder in which the BaseX code 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 located in a directory named by the name of the database. If the user's home directory has been chosen as base directory, the database directories will be planed in a `BaseXData` directory. Otherwise, the directory will be named `data`.

The database path can be changed as follows:

- GUI: Choose *Options* → *Preferences* and choose a new database path.
- General: edit the `DBPATH` option in the `.basex` configuration file

Note: Existing databases will not be automatically moved to the new destination.

Log Files

Log files are stored in text format in a `.logs` sub-directory of the database folder (see Logging for more information).

Changelog

Version 8.0

- Updated: `.basexperm` is obsolete. Users are now stored in `users.xml` in the database directory (see User Management for more information).

Version 7.7

- Updated: the `.basexhome` file marks a folder as home directory.

Chapter 94. Indexes

Read this entry online in the BaseX Wiki.

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 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. Furthermore, the index is enriched with statistical information, such as the distinct (categorical) or minimum and maximum values of its elements and attributes. The maximum number of categories to store per name can be changed via MAXCATS. The statistics are discarded after database updates and can be recreated with the OPTIMIZE command.

The name index is e.g. 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 with 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 contains the same statistical information as the name index. The statistics are discarded after database updates and can be recreated with the OPTIMIZE command.

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 with the XQuery function index:facets.

Resource Index

The resource index contains references to the `pre` values of all XML 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 resource 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

Exact Queries

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
```

Matching text nodes can be directly requested from the index with the XQuery function db:text. The index contents can be accessed via index:texts.

Range Queries

The text index also supports range queries based on string comparisons:

```
(: 1st example :)
db:open('Library')//Medium[Year >= '2011' and Year <= '2016'],
(: 2nd example :)
let $min := '2014-04-16T00:00:00'
let $max := '2014-04-19T23:59:59'
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.

Please note that the current index structures do not support queries for numbers and dates.

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']
```

Matching text nodes can be directly requested from the index with the XQuery functions db:attribute and db:attribute-range. The index contents can be accessed with 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]
```

Matching text nodes can be directly requested from the index via the XQuery function ft:search. The index contents can be accessed with ft:tokens.

Index Construction

If main memory runs out while creating a value index, the currently generated index structures will be partially written to disk and eventually merged. If the used memory heuristics fails for some reason (i.e., because multiple index operations run at the same time), fixed index split sizes may be chosen via the INDEXSPLITSIZE and FTINDEXSPLITSIZE options.

If DEBUG is set to true, and if a new database is created from command-line, the number of index operations will be output to standard output; this might help you to choose proper split size. The following example shows how the output can look like for a document with 111 MB and 128 MB of available main memory:

```
> basex -d -c"set ftindex; create db 111mb 111mb.xml"
Creating Database...
.... 8132.44 ms (17824 KB)
Indexing Text...
.. 979920 operations, 2913.78 ms (44 MB)
Indexing Attribute Values...
.. 381870 operations, 630.61 ms (21257 KB)
Indexing Full-Text...
..|| 3 splits, 12089347 operations, 16420.47 ms (36 MB)
```

The info string `3 splits` indicates that three partial full-text index structures were written to disk, and the string `12089347 operations` tells that the index construction consisted of appr. 12 mio index operations. If we set FTINDEXSPLITSIZE to the fixed value `4000000` (12 mio divided by three), or a smaller value, we should be able to build the index and circumvent the memory heuristics.

Updates

By default, index structures are discarded after an update operation. As a result, queries will be executed more slowly. There are different alternatives to cope with this:

After the execution of update operations, the OPTIMIZE command or the db:optimize function can be called to rebuild the index structures. This way, multiple update operations will be performed faster, as the database meta data is only updated and regenerated when requested by the database users.

With the UPDINDEX option, text and attributes index structures will incrementally be updated. This option must be turned on before the database is created or optimized. Please note that incremental updates are not available for the full-text index and database statistics. This is also the reason why the internal up-to-date flag of a database (which can e.g. be displayed via INFO DB or db:info) will be set to `false` until the next optimize call is triggered.

If AUTOOPTIMIZE is activated, all outdated index structures and statistics will be recreated after a database update. This option should only be activated for medium-sized databases. Similar to UPDINDEX, it must be turned on before the database is created or optimized.

Changelog

Version 8.0

- Added: AUTOOPTIMIZE option

Version 7.2.1

- Added: string-based range queries

Chapter 95. Backups

Read this entry online in the BaseX Wiki.

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.

Chapter 96. Catalog Resolver

Read this entry online in the BaseX Wiki.

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, available for download here.

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
- Apache XML Commons Article on Entity Resolving
- XML Entity and URI Resolvers , Sun
- XML Catalogs. OASIS Standard, Version 1.1. 07-October-2005.

Chapter 97. Storage Layout

Read this entry online in the BaseX Wiki.

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)
Num	Compressed integer (1-5 bytes), specified in Num.java	15 → 0F; 511 → 41 FF
Token	Length (*Num*) and bytes of UTF8 byte representation	Hello → 05 48 65 6c 6c 6f
Double	Number, stored as token	123 → 03 31 32 33
Boolean	Boolean (1 byte, 00 or 01)	true → 01
Nums , *Tokens* , *Doubles*	Arrays of values, introduced with the number of entries	1,2 → 02 01 31 01 32
TokenSet	Key array (*Tokens*), next/bucket/size arrays (3x *Nums*)	

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 (*Token* / *Token*): • Examples: FNAME, TIME, SIZE, ... • PERM → Number of users (*Num*), and name/password/permission values for each user (*Token* / *Token* / *Num*)2. Empty key as finalizer	DiskData() MetaData() Users()
2. Main memory indexes	1. Key/value pairs, in no particular order (*Token* / *Token*): • TAGS → Tag Index • ATTS → Attribute Name Index • PATH → Path Index • NS → Namespaces • DOCS → Document Index2. Empty key as finalizer	DiskData()
2 a) Name Index Tag/attribute names	1. Token set, storing all names (*TokenSet*)2. One StatsKey instance per entry:2.1. Content kind (*Num*):2.1.1. Number: min/max (*Doubles*)2.1.2. Category: number of entries (*Num*), entries (*Tokens*)2.2. Number of entries (*Num*)2.3. Leaf flag (*Boolean*)2.4.	Names() TokenSet.read() StatsKey()

	Maximum text length (*Double* ; legacy, could be *Num*)	
2 b) Path Index	1. Flag for path definition (*Boolean* , always `true`; legacy)2. PathNode: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 discarded2.6. Recursive generation of child nodes (→ 2)	PathSummary() PathNode()
2 c) Namespaces	1. Token set, storing prefixes (*TokenSet*)2. Token set, storing URIs (*TokenSet*)3. NSNode: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 (→ 3)	Namespaces() NSNode()
2 d) Document Index	Array of integers, representing the distances between all document pre values (*Nums*)	DocIndex()

Node Table: `tbl`, `tbli`

- `tbl` : Main database table, stored in blocks.
- `tbli` : Database directory, organizing the database blocks.

Some more information on the node 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.

For a more detailed discussion and examples of these file formats please see Index File Structure.

Full-Text Fuzzy Index: `ftxx`, `ftxy`, `ftxz`

...may soon be reimplemented.

Chapter 98. Node Storage

Read this entry online in the BaseX Wiki.

This article describes the Storage Layout of the main database table.

Node Table

BaseX stores all XML nodes in a flat table. The node table of a database can be displayed via the INFO STORAGE command:

```
$ basex -c"create db db <xml>HiThere</xml>" -c"info storage"

PRE  DIS  SIZ  ATS  ID  NS  KIND  CONTENT
----------------------------------------
  0    1    3    1   0   0  DOC   db.xml
  1    1    2    1   1   0  ELEM  xml
  2    1    1    1   2   0  TEXT  HiThere
```

PRE Value

The *pre* value of a node represents the order in which the XML nodes are visited by a SAX parser. It is actually not stored in the database; instead, it is implicitly given by the table position. As a result, it will change whenever a node with a smaller *pre* values is added to or deleted from a database.

ID Value

Each database node has a persistent *id* value, which remains valid after update operations, and which is referenced by the value indexes. As long as no updates are performed on a database, the *pre* and *id* values are identical. The values will remain to be identical if new nodes are exclusively added to the end of the database. If nodes are deleted or inserted somewhere else, the values will diverge, as shown in the next example:

```
$ basex -c"create db db <xml>HiThere</xml>" -q"insert node <b/> before /xml" -
c"info storage"

PRE  DIS  SIZ  ATS  ID  NS  KIND  CONTENT
----------------------------------------
  0    1    4    1   0   0  DOC   db.xml
  1    1    1    1   3   0  ELEM  b
  2    2    2    1   1   0  ELEM  xml
  3    1    1    1   2   0  TEXT  HiThere
```

The db:node-pre and db:node-id functions can be called to retrieve the *pre* and *id* values of a node, and db:open-pre and db:open-id can be used to go back and retrieve the original node. By default, *id* lookups are expensive. If the UPDINDEX option is turned on, an additional index will be maintained to speed up the process.

Block Storage

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 pres 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.

Chapter 99. User Management

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. The user management defines which permissions are required by a user to perform a database command or XQuery expression.

Permissions are mostly relevant in the client/server architecture, as the Standalone Mode and the GUI is run with admin permissions. There are a few exceptions such as the xquery:eval function: its execution scope can also be limited by specifying a permission.

With **Version 8.0**, the user management has been revised:

- Permissions can now be **manually edited**, as they are stored as XML.
- The permission file has been moved from the home directory to the **database directory**. It was renamed from `.basexperm` to `users.xml`.
- Local permissions are now defined for database **glob patterns** instead of single databases. Both local and global permissions are stored in the same file.
- A new User Module is available, which allows user management via **XQuery**.
- MD5 password hashing has been discarded, as many md5 hashes can be easily uncovered with rainbow tables.
- **Salted sha256** hashes are now used for authentication (the current timestamp will be used as salt).
- Additionally, **digest** hashes are used in the client/server architecture and the Language Bindings, and in the HTTP Context if the AUTHMETHOD is set to `Digest`.

Warning: As the available md5 hashes cannot automatically be converted to the new format, existing credentials will be ignored, and you will need to recreate your user data. Moreover, we will incrementally provide new Language Bindings, which will be based on the digest hashes.

Passwords in commands and XQuery functions are now specified in **plain text**. The rationale behind this is:

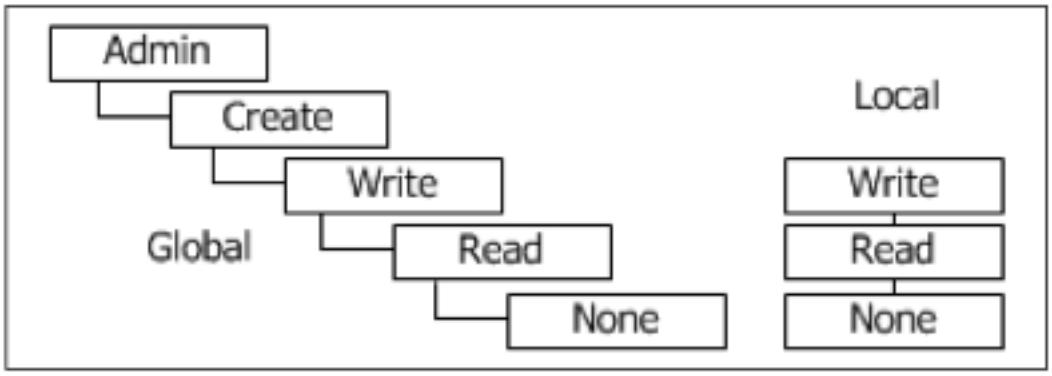
- Transmission of passwords as md5 has not been safe anyway, as indicated above.
- Different hash values can now be created from the original password.

You will be safe if you follow basic security measures: ensure that your password will not end up in your bash history, avoid sending passwords via ordinary REST requests, etc.

Rules

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.

Local permissions are applied to databases. They have a higher precedence and override global permissions.



Permissions hierarchy User names must follow the valid names constraints, and the database patterns must follow the Glob Syntax.

Commands

Admin permissions are required to execute all of the following commands:

Create user 'test' (password will be entered on command line). By default, the user will have no permissions ('none'):

```
> CREATE USER test
```

Change password of user 'test' to '71x343sd#':

```
> ALTER PASSWORD test 71x343sd#
```

Grant local write permissions to user 'test':

```
> GRANT write ON unit* 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) database starting with the letters 'unit'. If no local permissions are set, the global rights are inherited.

Show global permissions:

```
> SHOW USERS
```

Show detailed information about user 'test' via XQuery:

```
> XQUERY user:list-details()[@name = 'test']
```

Drop of user 'test' via XQuery:

```
> XQUERY user:drop('test')
```

Changelog

Revised in Version 8.0.

Chapter 100. Transaction Management

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. The BaseX client-server architecture offers ACID safe transactions, with multiple readers and writers. Here is some more information 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, may 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 try the INSPECT command to check if your database has obvious inconsistencies, and RESTORE to restore a previous version of the database.

Update Transactions

Many update operations are triggered by XQuery Update expressions. When executing an updating 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

BaseX provides locking on database level. Writing transactions do not necessarily block all other transactions any more. The number of parallel transactions can be limited by setting the PARALLEL option.

Transaction Monitor

The transaction monitor ensures that just one writing transaction or an arbitrary amount of reading transactions *per database* are active at the same time.

Deadlocks are prevented by using preclaiming two phase locking. Execution is starvation-free as lock acquisition is queued per database. Due to the specifics of XQuery Update, all updates are written at the end of the query. Locking is strict with the exception that databases for which BaseX recognizes it will not write to are downgraded to read locks.

Locks are not synchronized between multiple BaseX instances. We generally recommend working with the client/server architecture if concurrent write operations are to be performed.

External Side Effects

Access to external resources (files on hard disk, HTTP requests, ...) is not controlled by BaseX' transaction monitor unless specified by the user.

XQuery Locking Options

Custom locks can be acquired by setting the BaseX-specific XQuery options `query:read-lock` and `query:write-lock`. Multiple option declarations may occur in the prolog of a query, but multiple values can also be separated with commas in a single declaration. These locks are in another namespace than the database names: the lock value `factbook` will not lock a database named factbook.

These option declarations will put read locks on *foo*, *bar* and *batz* and a write lock on *quix*:

```
declare option query:read-lock "foo,bar";
declare option query:read-lock "batz";
declare option query:write-lock "quix";
```

Java Modules

Locks can also be acquired on Java functions which are imported and invoked from an XQuery expression. It is advisable to explicitly lock Java code whenever it performs sensitive read and write operations.

Limitations

Commands

Database locking works with all commands unless no glob syntax is used, such as in the following command call:

- `DROP DB new*` : drop all databases starting with "new"

XQuery

As XQuery is a very powerful language, deciding which databases will be accessed by a query is non-trivial. Optimization is work in progress. The current identification of which databases to lock is limited to queries that access the currently opened database, XQuery functions that explicitly specify a database, and expressions that address no database at all.

Some examples on database-locking enabled queries, all of these can be executed in parallel:

- `//item` , read-locking of the database opened by a client
- `doc('factbook')` , read-locking of "factbook"
- `collection('db/path/to/docs')` , read-locking of "db"
- `fn:sum(1 to 100)` , locking nothing at all
- `delete nodes doc('test')//*[string-length(local-name(.)) > 5]` , write-locking of "test"

Some examples on queries that are not supported by database-locking yet:

- `let $db := 'factbook' return doc($db)` , will read-lock: referencing database names isn't supported yet
- `for $db in ('factbook') return doc($db)` , will read-lock globally
- `doc(doc('test')/reference/text())` , will read-lock globally
- `let $db := 'test' return insert nodes <test/> into doc($db)` , will write-lock globally

A list of all locked databases is output if `QUERYINFO` is set to `true`. If you think that too much is locked, please give us a note on our mailing list with some example code.

GUI

Database locking is currently disabled if the BaseX GUI is used.

Process Locking

In order to enable locking on global (process) level, the option `GLOBALLOCK` can be set to `true`. This can e.g. be done by editing your `.basex` file (see Options for more details). If process locking is active, a process that performs write operations will queue all other operations.

File-System Locks

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

To avoid database corruptions caused by write operations running in different JVMs, 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.

As the standalone versions of BaseX (command-line, GUI) cannot be synchronized with other BaseX instances, we generally recommend working with the client/server architecture if concurrent write operations are to be performed.

Changelog

Version 7.8

- Added: Locks can also be acquired on Java functions.

Version 7.6

- Added: database locking introduced, replacing process locking.

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.

Chapter 101. Logging

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It describes how client operations are logged by the server. The server logs can e.g. be used to get an overview of all processes executed on your server, trace any errors or compile performance statistics.

The server logs are written in plain text. 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 beinning with a `.` may be hidden.

Some more notes on the logging facility:

- HTTP requests are included in the log files.
- Logging can be turned on/off via the LOG option.
- The maximum length of logging messages can be changed via LOGMSGMAXLEN.
- The Admin Module provides access to the log files from XQuery.

Format

Example 1

```
01:18:12.892   SERVER           admin   OK        Server was started (port: 1984)
01:18:15.436   127.0.0.1:4722   jack    REQUEST   XQUERY for $i in 1 to 5 return
 random:double()
01:18:15.446   127.0.0.1:4722   jack    OK        Query executed in 2.38 ms.
                2.72 ms
01:18:15.447   127.0.0.1:4722   jack    REQUEST   EXIT
01:18:15.447   127.0.0.1:4722   jack    OK
                0.39 ms
```

A server has been started and a user `jack` has connected to the server to perform a query and exit properly.

Example 2

```
01:23:33.251   127.0.0.1:4736   john   OK        QUERY[0] 'hi'   0.44 ms
01:23:33.337   127.0.0.1:4736   john   OK        ITER[0]         1.14 ms
01:23:33.338   127.0.0.1:4736   john   OK        INFO[0]         0.36 ms
01:23:33.339   127.0.0.1:4736   john   OK        CLOSE[0]        0.21 ms
01:23:33.359   127.0.0.1:4736   john   REQUEST   EXIT
01:23:33.359   127.0.0.1:4736   john   OK                        0.14 ms
```

A user `john` has performed an iterative query, using one of the client APIs.

Example 3

```
01:31:51.888   127.0.0.1:4803   admin   REQUEST   [GET] http://localhost:8984/rest/
factbook
01:31:51.892   127.0.0.1:4803   admin   200
           4.43 ms
```

An admin user has accessed the `factbook` database via REST.

Chapter 102. Execution Plan

Read this entry online in the BaseX Wiki.

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.

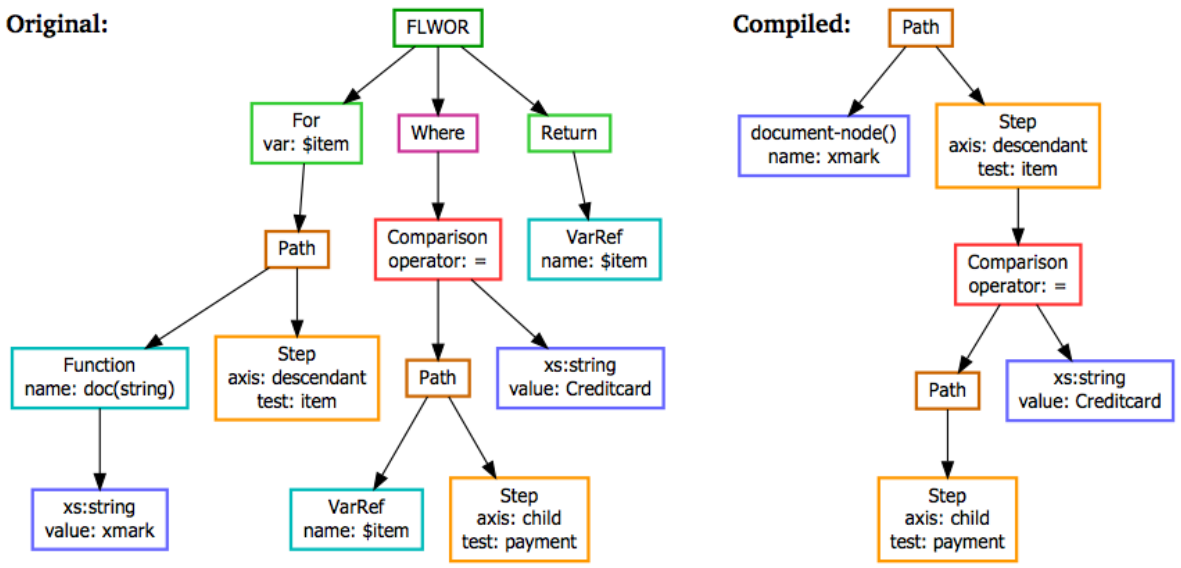
Examples

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:

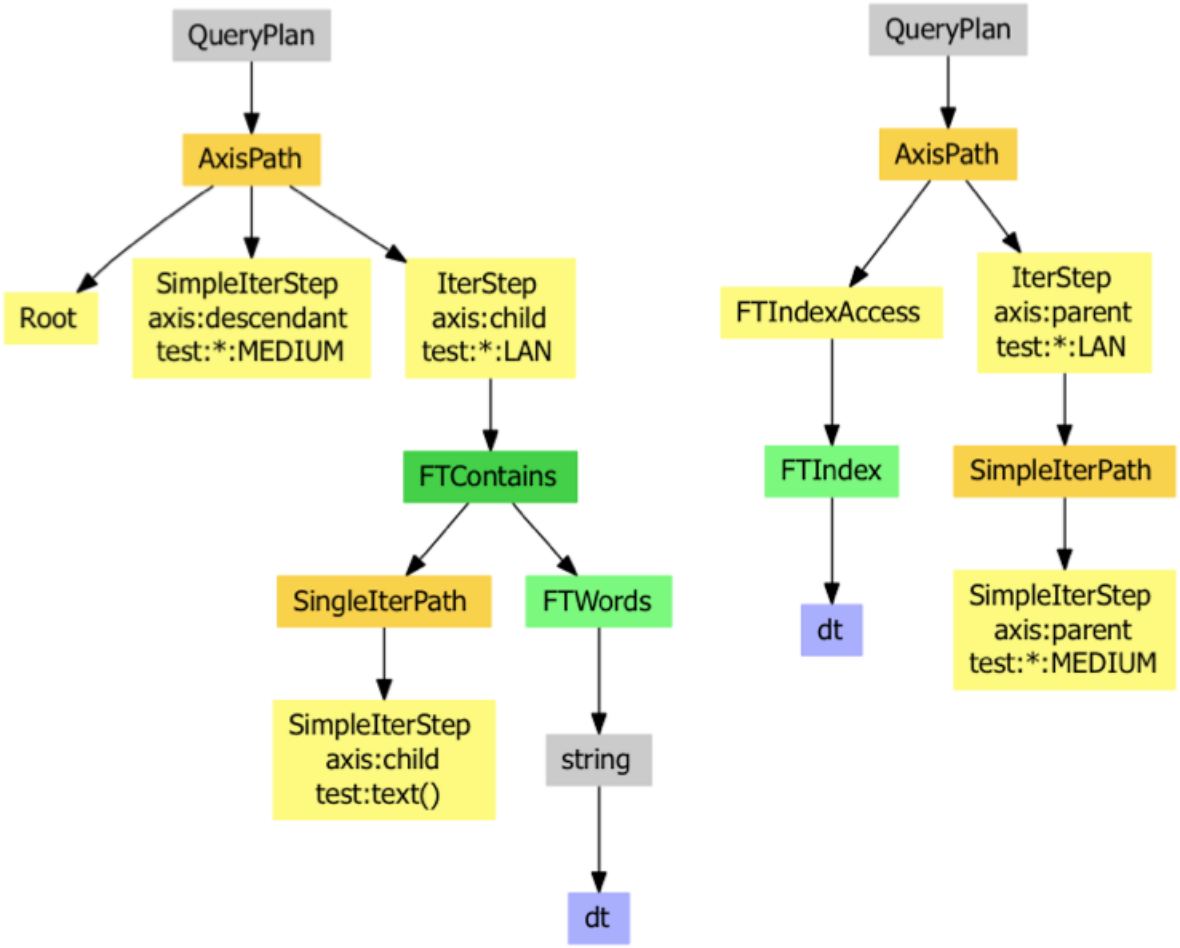


Replacing XQuery with equivalent XPath expressions

Execution plan for query execution with full-text index access and without

Query: `//MEDIUM/LAN[text() contains text "dt"]`

Execution plan:


QueryPlan
AxisPath
Root
SimpleIterStep
axis:descendant
test:*:MEDIUM
IterStep
axis:child
test:*:LAN
FTContains
SingleIterPath
FTWords
SimpleIterStep
axis:child
test:text()
string
dt
QueryPlan
AxisPath
FTIndexAccess
IterStep
axis:parent
test:*:LAN
FTIndex
SimpleIterPath
dt
SimpleIterStep
axis:parent
test:*:MEDIUM

Query Plan 2

Part XI. Use Cases

Chapter 103. Statistics

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It lists statistics on various XML instances that have been created with BaseX, with the value and full-text indexes turned off. 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 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)	*no limit*	**2'147'483'648**(2^31)	**648**(2^31)	**32768**(2^15)	**32768**(2^15)	**256**(2^8)	*no limit*
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	1'615'071'324	48	74	9	0	13
EnWikiMeta	54 GiB	1	52 GiB	401'456'348	3	21	6	2	6
MedLine	38 GiB	379	36 GiB	1'623'764'225	34	84	6	0	9
iProClass	36 GiB	1	37 GiB	1'631'218'984	34	245	4	2	9
Inex2009	31 GiB	2'666'500	34 GiB	1'336'110'639	39	28'034	451	1	37
CoPhIR	29 GiB	10'000'000	31 GiB	1'104'623'370	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
Genome1	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	3	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	http://air-climate.eionet.europa.eu/databases/airbase/airbasexml
Alfred	http://alfred.med.yale.edu/alfred/alfredWithDescription.zip
BibDBPub	http://inex.is.informatik.uni-duisburg.de/2005/
CoPhIR	http://cophir.isti.cnr.it/
DBLP	http://dblp.uni-trier.de/xml
DBLP2	http://inex.is.informatik.uni-duisburg.de/2005/
DDI	http://tools.ddialliance.org/
EnWikiMeta	http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-meta-current.xml.bz2
EnWikipedia	http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2
EnWikiRDF	http://www.xml-benchmark.org/ generated with xmlgen
EnWiktionary	http://dumps.wikimedia.org/enwiktionary/latest/enwiktionary-latest-pages-meta-history.xml.7z
EURLex	http://www.epsiplatform.eu/
Factbook	http://www.cs.washington.edu/research/xmldatasets/www/repository.html
Freebase	http://download.freebase.com/wex
FreeDB	http://www.xmldatabases.org/radio/xmlDatabases/projects/FreeDBtoXML
Freshmeat	http://freshmeat.net/articles/freshmeat-xml-rpc-api-available
Genome1	ftp://ftp.ncbi.nih.gov/snp/organisms/human_9606/XML/ds_ch1.xml.gz
HCIBIB2	http://inex.is.informatik.uni-duisburg.de/2005/
Inex2009	http://www.mpi-inf.mpg.de/departments/d5/software/inex
IntAct	ftp://ftp.ebi.ac.uk/pub/databases/intact/current/index.html
InterPro	ftp://ftp.bio.net/biomirror/interpro/match_complete.xml.gz
iProClass	ftp://ftp.pir.georgetown.edu/pir_databases/iproclass/iproclass.xml.gz
JMNEdict	ftp://ftp.monash.edu.au/pub/nihongo/enamdict_doc.html
KanjiDic2	http://www.csse.monash.edu.au/~jwb/kanjidic2
MedLine	http://www.nlm.nih.gov/bsd
MeSH	http://www.nlm.nih.gov/mesh/xmlmesh.html
MovieDB	http://eagereyes.org/InfoVisContest2007Data.html
MusicXML	http://www.recordare.com/xml/samples.html

Nasa	http://www.cs.washington.edu/research/xmldatasets/www/repository.html
NewYorkTimes	http://www.nytimes.com/ref/membercenter/nytarchive.html
OpenStreetMap	http://dump.wiki.openstreetmap.org/osmwiki-latest-files.tar.gz
Organizations	http://www.data.gov/raw/1358
RuWikiHist	http://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-meta-history.xml.7z
SDMX	http://www.metadatatechnology.com/
Shakespeare	http://www.cafeconleche.org/examples/shakespeare
SwissProt	ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase
Thesaurus	http://www.drze.de/BELIT/thesaurus
Treebank	http://www.cs.washington.edu/research/xmldatasets
TreeOfLife	http://tolweb.org/data/tolskeletaldump.xml
TrEMBL	ftp://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase
Wikicorpus	http://www-connex.lip6.fr/~denoyer/wikipediaXML
XMark	http://www.xml-benchmark.org/ generated with xmlgen
ZDNET	http://inex.is.informatik.uni-duisburg.de/2005/
ZhWikiHist	http://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-meta-history.xml.7z
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

Chapter 104. Twitter

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It is about the usage of BaseX for processing and storing the live data stream of Twitter. We illustrate some statistics about the Twitter data and the performance of BaseX.

As Twitter attracts more and more users (over 140 million active users in 2012) and is generating large amounts of data (over 340 millions of short messages ('tweets') daily), it became a really exciting data source for all kind of analytics. Twitter provides the developer community with a set of APIs for retrieving the data about its users and their communication, including the Streaming API for data-intensive applications, the Search API for querying and filtering the messaging content, and the REST API for accessing the core primitives of the Twitter platform.

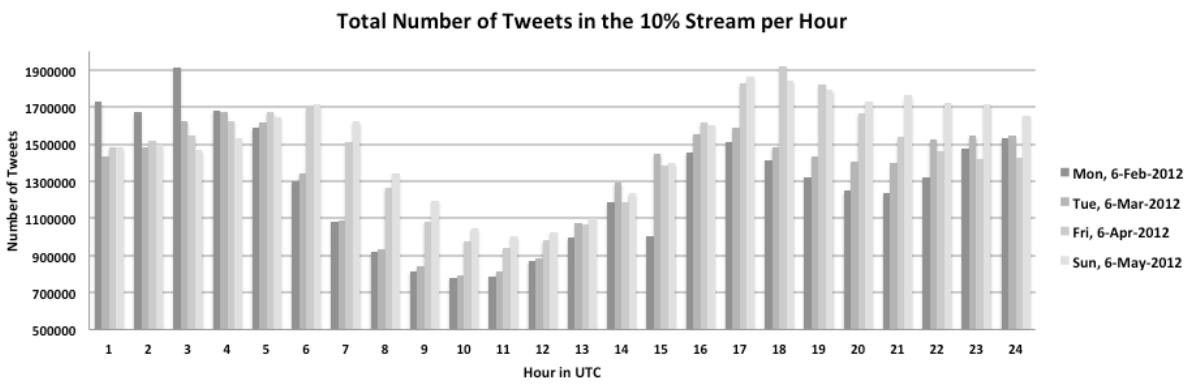
BaseX as Twitter Storage

For retrieving the Twitter stream we connect with the Streaming API to the endpoint of Twitter and receive a never ending tweet stream. As Twitter delivers the tweets as JSON objects the objects has to be converted into XML fragments. For this purpose the parse function of the XQuery JSON Module is used. In the examples section both versions are shown (tweet as JSON and tweet as XML). For storing the tweets including the meta-data, we use the standard *insert* function of XQuery Update.

Twitter's Streaming Data

Each tweet object in the data stream contains the tweet message itself and over 60 data fields (for further information see the fields description). The following section shows the amount of data, that is delivered by the Twitter Streaming API to the connected endpoints with the 10% gardenhose access per hour on the 6th of the months February, March, April and May. It is the pure public live stream without any filtering applied.

Statistics



Day	Description	Amount
Mon, 6-Feb-2012	Total tweets	30.824.976
	Average tweets per hour	1.284.374
	Average tweets per minute	21.406
	Average tweets per second	356
Tue, 6-Mar-2012	Total tweets	31.823.776
	Average tweets per hour	1.325.990
	Average tweets per minute	22.099
	Average tweets per second	368
Fri, 6-Apr-2012	Total tweets	34.638.976
	Average tweets per hour	1.443.290

	Average tweets per minute	24.054
	Average tweets per second	400
Sun, 6-May-2012	Total tweets	35.982.976
	Average tweets per hour	1.499.290
	Average tweets per minute	24.988
	Average tweets per second	416

Example Tweet (JSON)


```
{
  "contributors": null,
  "text": "Using BaseX for storing the Twitter Stream",
  "geo": null,
  "retweeted": false,
  "in_reply_to_screen_name": null,
  "possibly_sensitive": false,
  "truncated": false,
  "entities": {
    "urls": [
    ],
    "hashtags": [
    ],
    "user_mentions": [
    ]
  },
  "in_reply_to_status_id_str": null,
  "id": 1984009055807*****,
  "in_reply_to_user_id_str": null,
  "source": "<a href=\"http:\/\/twitterfeed.com\" rel=\"nofollow\">twitterfeed<\/
a>",
  "favorited": false,
  "in_reply_to_status_id": null,
  "retweet_count": 0,
  "created_at": "Fri May 04 13:17:16 +0000 2012",
  "in_reply_to_user_id": null,
  "possibly_sensitive_editable": true,
  "id_str": "1984009055807*****",
  "place": null,
  "user": {
    "location": "",
    "default_profile": true,
    "statuses_count": 9096,
    "profile_background_tile": false,
    "lang": "en",
    "profile_link_color": "0084B4",
    "id": 5024566**,
    "following": null,
    "protected": false,
    "favourites_count": 0,
    "profile_text_color": "333333",
    "contributors_enabled": false,
    "verified": false,
    "description": "http:\/\/basex.org",
    "profile_sidebar_border_color": "C0DEED",
    "name": "BaseX",
    "profile_background_color": "C0DEED",
    "created_at": "Sat Feb 25 04:05:30 +0000 2012",
    "default_profile_image": true,
    "followers_count": 860,
```

```
    "geo_enabled": false,
    "profile_image_url_https": "https:\/\/si0.twimg.com\/sticky\/
default_profile_images\/default_profile_0_normal.png",
    "profile_background_image_url": "http:\/\/a0.twimg.com\/images\/themes\/
theme1\/bg.png",
    "profile_background_image_url_https": "https:\/\/si0.twimg.com\/images\/themes
\/theme1\/bg.png",
    "follow_request_sent": null,
    "url": "http:\/\/adf.ly\/5ktAf",
    "utc_offset": null,
    "time_zone": null,
    "notifications": null,
    "friends_count": 2004,
    "profile_use_background_image": true,
    "profile_sidebar_fill_color": "DDEEF6",
    "screen_name": "BaseX",
    "id_str": "5024566**",
    "show_all_inline_media": false,
    "profile_image_url": "http:\/\/a0.twimg.com\/sticky\/default_profile_images\/
default_profile_0_normal.png",
    "is_translator": false,
    "listed_count": 0
  },
  "coordinates": null
}
```


Example Tweet (XML)


```
<json booleans="retweeted possibly__sensitive truncated favorited
 possibly__sensitive__editable default__profile profile__background__tile
 protected contributors__enabled verified default__profile__image geo__enabled
 profile__use__background__image show__all__inline__media is__translator"
  numbers="id retweet__count statuses__count favourites__count followers__count
 friends__count listed__count"
  nulls="contributors geo in__reply__to__screen__name
 in__reply__to__status__id__str in__reply__to__user__id__str
 in__reply__to__status__id in__reply__to__user__id place following
 follow__request__sent utc__offset time__zone notifications coordinates"
  arrays="urls indices hashtags user__mentions"
  objects="json entities user">
  <contributors/>
  <text>Using BaseX for storing the Twitter Stream</text>
  <geo/>
  <retweeted>false</retweeted>
  <in__reply__to__screen__name/>
  <possibly__sensitive>false</possibly__sensitive>
  <truncated>false</truncated>
  <entities>
    <urls/>
    <hashtags/>
    <user__mentions/>
  </entities>
  <in__reply__to__status__id__str/>
  <id>1984009055807*****</id>
  <in__reply__to__user__id__str/>
  <source><a href="http://twitterfeed.com" rel="nofollow">twitterfeed</a></source>
  <favorited>false</favorited>
  <in__reply__to__status__id/>
  <retweet__count>0</retweet__count>
  <created__at>Fri May 04 13:17:16 +0000 2012</created__at>
  <in__reply__to__user__id/>
  <possibly__sensitive__editable>true</possibly__sensitive__editable>
  <id__str>1984009055807*****</id__str>
  <place/>
```

```
  <user>
    <location/>
    <default__profile>true</default__profile>
    <statuses__count>9096</statuses__count>
    <profile__background__tile>false</profile__background__tile>
    <lang>en</lang>
    <profile__link__color>0084B4</profile__link__color>
    <id>5024566**</id>
    <following/>
    <protected>false</protected>
    <favourites__count>0</favourites__count>
    <profile__text__color>333333</profile__text__color>
    <contributors__enabled>false</contributors__enabled>
    <verified>false</verified>
    <description>http://basex.org</description>
    <profile__sidebar__border__color>C0DEED</profile__sidebar__border__color>
    <name>BaseX</name>
    <profile__background__color>C0DEED</profile__background__color>
    <created__at>Sat Feb 25 04:05:30 +0000 2012</created__at>
    <default__profile__image>true</default__profile__image>
    <followers__count>860</followers__count>
    <geo__enabled>false</geo__enabled>
    <profile__image__url__https>https://si0.twimg.com/sticky/
default_profile_images/default_profile_0_normal.png</profile__image__url__https>
    <profile__background__image__url>http://a0.twimg.com/images/themes/theme1/
bg.png</profile__background__image__url>
    <profile__background__image__url__https>https://si0.twimg.com/images/themes/
theme1/bg.png</profile__background__image__url__https>
    <follow__request__sent/>
    <url>http://adf.ly/5ktAf</url>
    <utc__offset/>
    <time__zone/>
    <notifications/>
    <friends__count>2004</friends__count>
    <profile__use__background__image>true</profile__use__background__image>
    <profile__sidebar__fill__color>DDEEF6</profile__sidebar__fill__color>
    <screen__name>BaseX</screen__name>
    <id__str>5024566**</id__str>
    <show__all__inline__media>false</show__all__inline__media>
    <profile__image__url>http://a0.twimg.com/sticky/default_profile_images/
default_profile_0_normal.png</profile__image__url>
    <is__translator>false</is__translator>
    <listed__count>0</listed__count>
  </user>
  <coordinates/>
</json>
```


BaseX Performance

The test show the time BaseX needs to insert large amounts of real tweets into a database. We can derive that BaseX scales very well and can keep up with the incoming amount of tweets in the stream. Some lower values can occur, cause the size of the tweets differ according to the meta-data contained in the tweet object. Note: The `AUTOFLUSH` option is set to `FALSE` (default: `SET AUTOFLUSH TRUE`)

System Setup: Mac OS X 10.6.8, 3.2 GHz Intel Core i3, 8 GB 1333 MHz DDR3 RAM BaseX Version: BaseX 7.3 beta

Insert with XQuery Update

These tests show the performance of BaseX performing inserts with XQuery Update as single updates per tweet or bulk updates with different amount of tweets. The initial database just contained a root node `<tweets/>` and all incoming tweets are inserted after converting from JSON to XML into the root node. The time needed for the inserts includes the conversion time.

Single Updates

Amount of tweets	Time in seconds	Time in minutes	Database Size (without indexes)
1.000.000	492.26346	8.2	3396 MB
2.000.000	461.87326	7.6	6997 MB
3.000.000	470.7054	7.8	10452 MB