BaseX Documentation

Version 10
# Table of Contents

I. Main Page

1. Main Page

   Getting Started .......................................................... 2
   XQuery Portal ............................................................ 2
   Advanced User's Guide .................................................. 3

2. BaseX 10 ........................................................................ 4

   Prerequisites ...................................................................... 4
   Migrating Applications .................................................... 4
   Storage .............................................................................. 4

      Whitespaces ................................................................... 4
      Value Resources ........................................................... 4
      Backups .......................................................................... 5
   XQuery .............................................................................. 5

      Compilation .................................................................... 5
      Main-Memory Updates ................................................... 6
      Key-Value Store ............................................................ 6
      Modules .......................................................................... 6
   Commands ......................................................................... 7
   HTTP Requests ................................................................... 7
   Catalogs ............................................................................ 7
   Graphical User Interface .................................................. 7
   REST ................................................................................ 8

II. Getting Started .................................................................. 9

3. Getting Started

   Overview ............................................................................ 10
   Tutorials and Slides .......................................................... 10

4. Startup ............................................................................... 12

   Introduction ....................................................................... 12
   Startup .............................................................................. 12

      Core Package .................................................................. 12
      Full Distributions .......................................................... 12
      Web Archive ................................................................... 13
      Other Distributions ........................................................ 13
   Concurrent Operations ..................................................... 13
   Changelog ......................................................................... 13

5. Graphical User Interface ................................................... 15

   Startup ............................................................................... 15
   Introduction ...................................................................... 15
   Database Management ..................................................... 16
   Editor ............................................................................... 17

      Project View ................................................................... 18
   Input Bar .......................................................................... 18
      Find .............................................................................. 18
      XQuery ........................................................................... 18
      Command ...................................................................... 18
   Visualizations ................................................................. 19
   Real-time Options ........................................................... 20
   Look and Feel .................................................................. 20
   Changelog ........................................................................ 20

6. Shortcuts .......................................................................... 22

   Editor ............................................................................... 22

      Code Completions .......................................................... 22
      Editor Shortcuts ............................................................. 22
   Changelog ........................................................................ 25

7. Command-Line Client ....................................................... 26
III. Command Line ................................................................. 37
  11. Command-Line Options .................................................. 38
      Standalone .................................................................. 38
      GUI ......................................................................... 40
      Server ..................................................................... 40
      Client ....................................................................... 40
      HTTP Server ............................................................ 42
      Changelog .................................................................... 43
  12. Start Scripts ................................................................. 45
      Standalone .................................................................. 45
          Windows: baseX.bat ................................................. 45
          Linux/Mac: baseX .................................................. 45
      GUI, Server, Client .................................................... 46
      HTTP Server ............................................................ 46
          Windows: baseXhttp.bat ......................................... 46
          Linux/Mac: baseXhttp ........................................... 46
      Included Start Scripts ................................................ 46
      Changelog .................................................................... 47

IV. General Info ................................................................. 48
  13. Databases .................................................................. 49
      Create Databases ........................................................ 49
      Access Resources ........................................................ 49
          XML Documents ...................................................... 49
          Binary Data ............................................................ 50
          XQuery Values ....................................................... 51
SHOW SESSIONS ................................................................. 63
SHOW USERS ............................................................... 64
KILL ................................................................. 64
INFO DB ......................................................... 64
INFO INDEX ...................................................... 64
INFO STORAGE .................................................. 64
General Commands .................................................. 65
RUN ......................................................... 65
EXECUTE .................................................... 65
SET ............................................................. 65
INFO ............................................................. 65
HELP ............................................................. 65
EXIT ............................................................... 66
QUIT ............................................................... 66
Changelog ................................................................ 66
15. Options ................................................................. 69
Global Options .......................................................... 69
General Options ..................................................... 70
Client/Server Architecture ........................................... 71
HTTP Services .......................................................... 74
Create Options .......................................................... 75
General ............................................................. 75
Parsing ............................................................. 76
XML Parsing .......................................................... 77
Indexing ............................................................. 79
Full-Text Indexing .................................................... 81
Query Options .......................................................... 82
QUERYINFO ..................................................... 82
MIXUPDATES ..................................................... 82
BINDINGS .......................................................... 82
INLINELIMIT ..................................................... 83
UNROLLLIMIT ..................................................... 83
ENFORCEINDEX .................................................. 83
COPYNODE ......................................................... 83
TAILCALLS .......................................................... 84
WITHDB ............................................................. 84
DEFAULTDB ......................................................... 84
FORCECREATE ..................................................... 84
CHECKSTRINGS ..................................................... 84
WRAPJAVA .......................................................... 85
LSERROR ............................................................ 85
RUNQUERY .......................................................... 85
RUNS ............................................................... 85
Serialization Options .................................................. 85
SERIALIZE .......................................................... 85
SERIALIZER .......................................................... 85
EXPORTER .......................................................... 86
XMLPLAN ........................................................... 86
FULLPLAN ........................................................... 86
Other Options .......................................................... 86
AUTOFLUSH .......................................................... 86
WRITEBACK .......................................................... 86
MAXSTAT ............................................................ 87
Changelog ................................................................ 87
16. Configuration .......................................................... 90
Configuration Files .................................................. 90
Home Directory .......................................................... 90
Database Directory .................................................. 90
Log Files ................................................................. 90
Changelog ............................................................. 91
V. Integration .......................................................... 92
17. Integrating oXygen ................................................. 93
   Access Database Resources ...................................... 93
      Preparations .................................................... 93
      Configuration .................................................. 93
   Perform Queries .................................................. 94
      One-Time Setup ............................................... 94
      Execute Query ............................................... 95
18. Integrating Eclipse ............................................... 96
   Preparations ...................................................... 96
   Access Database Resources .................................... 96
   Preparations ...................................................... 96
   Configuration .................................................... 96
   Perform Queries .................................................. 97
      One-Time Setup ............................................... 97
      Execute Query ............................................... 98
19. Integrating Intellij IDEA .......................................... 99
   Preparations ...................................................... 99
   xquery-intellij-plugin .......................................... 99
      Installation .................................................... 99
      Configuring The Processor ................................... 100
      Querying Your Data ......................................... 100
      Conclusion .................................................... 102
   XQuery Support Plugin ......................................... 102
      Installation .................................................... 102
      Setting Up ..................................................... 103
      Configuring The Processor ................................... 103
      Querying Your Data ......................................... 105
      Conclusion .................................................... 106
VI. XQuery Portal ................................................... 107
20. XQuery .............................................................. 108
21. XQuery 3.0 .......................................................... 109
   Enhanced FLWOR Expressions ................................... 109
      group by ......................................................... 109
      count ......................................................... 110
      allowing empty ............................................. 110
      window ....................................................... 110
   Function Items .................................................. 111
   Simple Map Operator ........................................... 111
   Try/Catch .......................................................... 111
   Switch ............................................................. 112
   Expanded QNames ............................................... 112
   Namespace Constructors ....................................... 113
   String Concatenations .......................................... 113
   External Variables .............................................. 113
   Serialization ..................................................... 113
   Context Item .................................................... 113
   Annotations ...................................................... 114
   Functions ......................................................... 114
   Changelog ........................................................ 114
22. Higher-Order Functions ......................................... 116
   Function Items .................................................. 116
      Function Types ............................................... 116
      Higher-Order Functions ...................................... 116
      Sequences ...................................................... 117
      Folds ........................................................... 119
23. XQuery 3.1 ........................................................................................................ 122
   Maps ........................................................................................................ 122
   Arrays ..................................................................................................... 122
      Atomization .................................................................................... 123
   Lookup Operator ............................................................................... 123
   Arrow Operator ................................................................................ 124
   String Constructor ........................................................................... 124
   Serialization ...................................................................................... 125
      Adaptive Serialization .................................................................... 125
      JSON Serialization ......................................................................... 125
   Functions ............................................................................................ 125
      Map Functions ................................................................................ 125
      Array Functions ............................................................................ 126
      JSON Functions ........................................................................... 126
      fn:sort ............................................................................................. 127
      fn:contains-token ......................................................................... 127
      fn:parse-ietf-date ......................................................................... 127
      fn:apply .......................................................................................... 127
      fn:random-number-generator ..................................................... 128
      fn:format-number ......................................................................... 128
      fn:tokenize ..................................................................................... 128
      fn:trace .......................................................................................... 128
      fn:string-join ................................................................................ 128
      fn:default-language ..................................................................... 129
   Appendix ................................................................................................ 129
   Binary Data .......................................................................................... 129
   Collations ............................................................................................ 129
   Enclosed Expressions ........................................................................ 129
   Changelog ............................................................................................ 129
24. XQuery Extensions .................................................................................. 131
   Expressions .......................................................................................... 131
      Ternary If ........................................................................................ 131
      Elvis Operator ................................................................................ 131
      If Without Else ............................................................................... 131
   Functions ............................................................................................ 132
      Regular Expressions ..................................................................... 132
   Serialization ........................................................................................ 132
   Option Declarations .......................................................................... 132
      Database Options ........................................................................... 132
      XQuery Locks ................................................................................ 132
   Pragmas ............................................................................................... 132
      BaseX Pragmas ............................................................................. 132
      Database Pragmas ........................................................................ 133
   Annotations ........................................................................................ 133
      Function Inlining ............................................................................ 133
      Lazy Evaluation ............................................................................. 134
      XQuery Locks ................................................................................ 134
   Non-Determinism .............................................................................. 134
   Namespaces ........................................................................................ 135
   Suffixes ............................................................................................... 135
   Miscellaneous ..................................................................................... 135
   Changelog ............................................................................................ 135
25. XQuery Optimizations ............................................................................ 136
   Introduction ........................................................................................ 136
   Compilation ........................................................................................ 136
      Pre-Evaluation ............................................................................... 136
      Variable Inlining ............................................................................ 136
      Function Inlining ............................................................................ 137
BaseX Documentation

Loop Unrolling ................................................................. 137
Paths .............................................................. 138
FLWOR Rewritings ................................................ 138
Static Typing .............................................................. 139
Pure Logic ............................................................... 140
Optimization ............................................................ 140
Database Statistics .................................................. 140
Index Rewritings .................................................... 141
Evaluation ............................................................... 143
Comparisons .............................................................. 143
Changelog ................................................................. 143
26. Module Library ................................................................. 144
Conventions .............................................................. 144
Changelog ................................................................. 146
27. Java Bindings ................................................................. 147
Identification ............................................................. 147
Classes ............................................................... 147
Functions and Variables ........................................ 147
Namespace Declarations ........................................... 148
Module Imports ................................................................. 149
Integration ................................................................. 149
Annotations .............................................................. 149
Updates ................................................................. 150
Locking ................................................................. 151
Data Types .............................................................. 151
URI Rewriting ............................................................. 153
Changelog ................................................................. 154
28. Repository ................................................................. 155
Introduction ................................................................. 155
Accessing Modules .................................................. 155
Commands ................................................................. 155
Installation .............................................................. 156
Listing ................................................................. 156
Removal ................................................................. 156
Packaging ................................................................. 156
XQuery ................................................................. 156
Java ................................................................. 156
Combined ............................................................ 158
EXPath Packaging .................................................... 159
XQuery ................................................................. 159
Java ................................................................. 159
Performance ............................................................. 160
Changelog ................................................................. 160
29. Full-Text ................................................................. 161
Introduction ................................................................. 161
Combining Results .................................................. 161
Positional Filters ..................................................... 162
Match Options .............................................................. 162
BaseX Features ................................................................. 163
Languages .............................................................. 163
Scoring ................................................................. 164
Thesaurus .............................................................. 164
Fuzzy Querying .............................................................. 165
Mixed Content .......................................................... 165
Functions .............................................................. 166
Collations .............................................................. 166
Changelog ................................................................. 167
30. XQuery Update ................................................................. 168
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Features</td>
<td>168</td>
</tr>
<tr>
<td>Updating Expressions</td>
<td>168</td>
</tr>
<tr>
<td>Main-Memory Updates</td>
<td>169</td>
</tr>
<tr>
<td>Functions</td>
<td>170</td>
</tr>
<tr>
<td>VII. XQuery Modules</td>
<td>196</td>
</tr>
<tr>
<td>Concepts</td>
<td>171</td>
</tr>
<tr>
<td>Pending Update List</td>
<td>171</td>
</tr>
<tr>
<td>Returning Results</td>
<td>171</td>
</tr>
<tr>
<td>Effects</td>
<td>172</td>
</tr>
<tr>
<td>Original Files</td>
<td>172</td>
</tr>
<tr>
<td>Indexes</td>
<td>172</td>
</tr>
<tr>
<td>Error Messages</td>
<td>172</td>
</tr>
<tr>
<td>Changelog</td>
<td>172</td>
</tr>
<tr>
<td>31. Indexes</td>
<td>174</td>
</tr>
<tr>
<td>Structural Indexes</td>
<td>174</td>
</tr>
<tr>
<td>Name Index</td>
<td>174</td>
</tr>
<tr>
<td>Path Index</td>
<td>174</td>
</tr>
<tr>
<td>Document Index</td>
<td>175</td>
</tr>
<tr>
<td>Value Indexes</td>
<td>175</td>
</tr>
<tr>
<td>Text Index</td>
<td>175</td>
</tr>
<tr>
<td>Attribute Index</td>
<td>176</td>
</tr>
<tr>
<td>Token Index</td>
<td>176</td>
</tr>
<tr>
<td>Full-Text Index</td>
<td>177</td>
</tr>
<tr>
<td>Selective Indexing</td>
<td>177</td>
</tr>
<tr>
<td>Enforce Rewritings</td>
<td>178</td>
</tr>
<tr>
<td>Custom Index Structures</td>
<td>179</td>
</tr>
<tr>
<td>Performance</td>
<td>179</td>
</tr>
<tr>
<td>Updates</td>
<td>180</td>
</tr>
<tr>
<td>Changelog</td>
<td>180</td>
</tr>
<tr>
<td>32. Serialization</td>
<td>182</td>
</tr>
<tr>
<td>Parameters</td>
<td>182</td>
</tr>
<tr>
<td>Character mappings</td>
<td>185</td>
</tr>
<tr>
<td>Changelog</td>
<td>185</td>
</tr>
<tr>
<td>33. XQuery Errors</td>
<td>187</td>
</tr>
<tr>
<td>Static Errors</td>
<td>187</td>
</tr>
<tr>
<td>Type Errors</td>
<td>189</td>
</tr>
<tr>
<td>Dynamic Errors</td>
<td>189</td>
</tr>
<tr>
<td>Functions Errors</td>
<td>190</td>
</tr>
<tr>
<td>Serialization Errors</td>
<td>192</td>
</tr>
<tr>
<td>Update Errors</td>
<td>192</td>
</tr>
<tr>
<td>Full-Text Errors</td>
<td>194</td>
</tr>
<tr>
<td>BaseX Errors</td>
<td>194</td>
</tr>
<tr>
<td>VII. XQuery Modules</td>
<td>196</td>
</tr>
<tr>
<td>34. Admin Module</td>
<td>197</td>
</tr>
<tr>
<td>Conventions</td>
<td>197</td>
</tr>
<tr>
<td>Database Logs</td>
<td>197</td>
</tr>
<tr>
<td>admin:logs</td>
<td>197</td>
</tr>
<tr>
<td>admin:write-log</td>
<td>197</td>
</tr>
<tr>
<td>admin:delete-logs</td>
<td>197</td>
</tr>
<tr>
<td>Database Sessions</td>
<td>197</td>
</tr>
<tr>
<td>admin:sessions</td>
<td>197</td>
</tr>
<tr>
<td>Errors</td>
<td>198</td>
</tr>
<tr>
<td>Changelog</td>
<td>198</td>
</tr>
<tr>
<td>35. Archive Module</td>
<td>199</td>
</tr>
<tr>
<td>Conventions</td>
<td>199</td>
</tr>
<tr>
<td>Content Handling</td>
<td>199</td>
</tr>
<tr>
<td>archive:entries</td>
<td>199</td>
</tr>
<tr>
<td>archive:options</td>
<td>199</td>
</tr>
<tr>
<td>archive:extract-text</td>
<td>199</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>archive:extract-binary</td>
<td>200</td>
</tr>
<tr>
<td>Updates</td>
<td>200</td>
</tr>
<tr>
<td>archive:create</td>
<td>200</td>
</tr>
<tr>
<td>archive:read</td>
<td>200</td>
</tr>
<tr>
<td>archive:write</td>
<td>201</td>
</tr>
<tr>
<td>Convenience</td>
<td>202</td>
</tr>
<tr>
<td>archive:create-from</td>
<td>202</td>
</tr>
<tr>
<td>archive:extract-to</td>
<td>202</td>
</tr>
<tr>
<td>archive:write</td>
<td>202</td>
</tr>
<tr>
<td>Errors</td>
<td>203</td>
</tr>
<tr>
<td>Changelog</td>
<td>203</td>
</tr>
<tr>
<td>Array Module</td>
<td>204</td>
</tr>
<tr>
<td>Conventions</td>
<td>204</td>
</tr>
<tr>
<td>Functions</td>
<td>204</td>
</tr>
<tr>
<td>array:size</td>
<td>204</td>
</tr>
<tr>
<td>array:get</td>
<td>204</td>
</tr>
<tr>
<td>array:append</td>
<td>204</td>
</tr>
<tr>
<td>array:subarray</td>
<td>204</td>
</tr>
<tr>
<td>array:put</td>
<td>204</td>
</tr>
<tr>
<td>array:remove</td>
<td>205</td>
</tr>
<tr>
<td>array:insert-before</td>
<td>205</td>
</tr>
<tr>
<td>array:head</td>
<td>205</td>
</tr>
<tr>
<td>array:tail</td>
<td>205</td>
</tr>
<tr>
<td>array:reverse</td>
<td>205</td>
</tr>
<tr>
<td>array:join</td>
<td>205</td>
</tr>
<tr>
<td>array:flatten</td>
<td>206</td>
</tr>
<tr>
<td>array:for-each</td>
<td>206</td>
</tr>
<tr>
<td>array:filter</td>
<td>206</td>
</tr>
<tr>
<td>array:fold-left</td>
<td>206</td>
</tr>
<tr>
<td>array:fold-right</td>
<td>206</td>
</tr>
<tr>
<td>array:for-each-pair</td>
<td>207</td>
</tr>
<tr>
<td>array:sort</td>
<td>207</td>
</tr>
<tr>
<td>Errors</td>
<td>207</td>
</tr>
<tr>
<td>Changelog</td>
<td>207</td>
</tr>
<tr>
<td>Binary Module</td>
<td>209</td>
</tr>
<tr>
<td>Conventions</td>
<td>209</td>
</tr>
<tr>
<td>Constants and Conversions</td>
<td>209</td>
</tr>
<tr>
<td>bin:hex</td>
<td>209</td>
</tr>
<tr>
<td>bin:bin</td>
<td>209</td>
</tr>
<tr>
<td>bin:octal</td>
<td>209</td>
</tr>
<tr>
<td>bin:to-octets</td>
<td>210</td>
</tr>
<tr>
<td>bin:from-octets</td>
<td>210</td>
</tr>
<tr>
<td>Basic Operations</td>
<td>210</td>
</tr>
<tr>
<td>bin:length</td>
<td>210</td>
</tr>
<tr>
<td>bin:part</td>
<td>210</td>
</tr>
<tr>
<td>bin:join</td>
<td>210</td>
</tr>
<tr>
<td>bin:insert-before</td>
<td>210</td>
</tr>
<tr>
<td>bin:pad-left</td>
<td>210</td>
</tr>
<tr>
<td>bin:pad-right</td>
<td>211</td>
</tr>
<tr>
<td>bin:find</td>
<td>211</td>
</tr>
<tr>
<td>Text Decoding and Encoding</td>
<td>211</td>
</tr>
<tr>
<td>bin:decode-string</td>
<td>211</td>
</tr>
<tr>
<td>bin:encode-string</td>
<td>211</td>
</tr>
<tr>
<td>Packing and Unpacking of Numeric Values</td>
<td>212</td>
</tr>
<tr>
<td>bin:pack-double</td>
<td>212</td>
</tr>
<tr>
<td>bin:pack-float</td>
<td>212</td>
</tr>
<tr>
<td>bin:pack-integer</td>
<td>212</td>
</tr>
<tr>
<td>bin:unpack-double</td>
<td>212</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Changelog</td>
<td>214</td>
</tr>
<tr>
<td>Errors</td>
<td>214</td>
</tr>
<tr>
<td>Client Module</td>
<td>215</td>
</tr>
<tr>
<td>Conventions</td>
<td>215</td>
</tr>
<tr>
<td>Functions</td>
<td>215</td>
</tr>
<tr>
<td>Bitwise Operations</td>
<td>213</td>
</tr>
<tr>
<td>bin:or</td>
<td>213</td>
</tr>
<tr>
<td>bin:xor</td>
<td>213</td>
</tr>
<tr>
<td>bin:and</td>
<td>214</td>
</tr>
<tr>
<td>bin:not</td>
<td>214</td>
</tr>
<tr>
<td>bin:shift</td>
<td>214</td>
</tr>
<tr>
<td>Errors</td>
<td>216</td>
</tr>
<tr>
<td>Changelog</td>
<td>217</td>
</tr>
<tr>
<td>Conversion Module</td>
<td>218</td>
</tr>
<tr>
<td>Conventions</td>
<td>218</td>
</tr>
<tr>
<td>Strings</td>
<td>218</td>
</tr>
<tr>
<td>convert:binary-to-string</td>
<td>218</td>
</tr>
<tr>
<td>convert:string-to-base64</td>
<td>218</td>
</tr>
<tr>
<td>convert:string-to-hex</td>
<td>218</td>
</tr>
<tr>
<td>Binary Data</td>
<td>219</td>
</tr>
<tr>
<td>convert:integers-to-base64</td>
<td>219</td>
</tr>
<tr>
<td>convert:integers-to-hex</td>
<td>219</td>
</tr>
<tr>
<td>convert:binary-to-integers</td>
<td>219</td>
</tr>
<tr>
<td>convert:binary-to-bytes</td>
<td>219</td>
</tr>
<tr>
<td>Numbers</td>
<td>219</td>
</tr>
<tr>
<td>convert:integer-to-base</td>
<td>219</td>
</tr>
<tr>
<td>convert:integer-from-base</td>
<td>220</td>
</tr>
<tr>
<td>Dates and Durations</td>
<td>220</td>
</tr>
<tr>
<td>convert:integer-to-dateTime</td>
<td>220</td>
</tr>
<tr>
<td>convert:dateTime-to-integer</td>
<td>220</td>
</tr>
<tr>
<td>convert:integer-to-dayTime</td>
<td>220</td>
</tr>
<tr>
<td>convert:dayTime-to-integer</td>
<td>221</td>
</tr>
<tr>
<td>Keys</td>
<td>221</td>
</tr>
<tr>
<td>convert:encode-key</td>
<td>221</td>
</tr>
<tr>
<td>convert:decode-key</td>
<td>221</td>
</tr>
<tr>
<td>Errors</td>
<td>221</td>
</tr>
<tr>
<td>Changelog</td>
<td>222</td>
</tr>
<tr>
<td>Cryptographic Module</td>
<td>223</td>
</tr>
<tr>
<td>Conventions</td>
<td>223</td>
</tr>
<tr>
<td>Message Authentication</td>
<td>223</td>
</tr>
<tr>
<td>crypto:hmac</td>
<td>223</td>
</tr>
<tr>
<td>Encryption &amp; Decryption</td>
<td>223</td>
</tr>
<tr>
<td>crypto:encrypt</td>
<td>224</td>
</tr>
<tr>
<td>crypto:decrypt</td>
<td>224</td>
</tr>
<tr>
<td>XML Signatures</td>
<td>224</td>
</tr>
<tr>
<td>crypto:generate-signature</td>
<td>225</td>
</tr>
<tr>
<td>crypto:validate-signature</td>
<td>226</td>
</tr>
<tr>
<td>Errors</td>
<td>227</td>
</tr>
<tr>
<td>Changelog</td>
<td>227</td>
</tr>
<tr>
<td>CSV Module</td>
<td>229</td>
</tr>
</tbody>
</table>
Conventions ........................................................................................................ 229
Conversion ........................................................................................................ 229
Options .............................................................................................................. 230
Functions .......................................................................................................... 232
csv:doc ............................................................................................................. 232
csv:parse .......................................................................................................... 232
csv:serialize ................................................................................................. 233
Examples .......................................................................................................... 233
Errors ............................................................................................................... 234
Changelog ........................................................................................................ 234
42. Database Module ........................................................................................ 235
Conventions ....................................................................................................... 235
  Database Nodes ............................................................................................ 235
  Updating Functions ...................................................................................... 235
General Functions ............................................................................................. 235
db:system ........................................................................................................ 235
db:option ......................................................................................................... 235
db:info ............................................................................................................ 236
db:property ...................................................................................................... 236
db:list ............................................................................................................... 236
db:list-details ................................................................................................. 236
db:dir ............................................................................................................... 237
Read Operations ................................................................................................. 237
db:get .............................................................................................................. 237
db:get-pre ....................................................................................................... 237
db:get-id ......................................................................................................... 238
db:get-binary .................................................................................................. 238
db:get-value .................................................................................................... 238
db:node-pre ..................................................................................................... 238
db:node-id ....................................................................................................... 239
db:export ......................................................................................................... 239
Value Indexes ..................................................................................................... 239
db:text .............................................................................................................. 239
db:text-range .................................................................................................. 240
db:attribute ..................................................................................................... 240
db:attribute-range ......................................................................................... 240
db:token ......................................................................................................... 240
Updates ............................................................................................................... 241
db:create ......................................................................................................... 241
db:add .............................................................................................................. 241
db:put .............................................................................................................. 242
db:put-binary ................................................................................................. 243
db:put-value ................................................................................................. 243
db:delete ......................................................................................................... 243
db:copy .......................................................................................................... 244
db:alter .......................................................................................................... 244
db:optimize .................................................................................................... 244
db:rename ...................................................................................................... 244
db:flush ......................................................................................................... 245
db:drop .......................................................................................................... 245
Backups ............................................................................................................. 245
db:create-backup ......................................................................................... 245
db:drop-backup .............................................................................................. 245
db:alter-backup .............................................................................................. 246
db:restore ....................................................................................................... 246
db:backups ................................................................................................. 246
Helper Functions ............................................................................................... 246
db:name ......................................................................................................... 246
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>db:path</code></td>
<td>246</td>
</tr>
<tr>
<td><code>db:exists</code></td>
<td>246</td>
</tr>
<tr>
<td><code>db:type</code></td>
<td>247</td>
</tr>
<tr>
<td><code>db:content-type</code></td>
<td>247</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Changelog</strong></td>
<td></td>
</tr>
<tr>
<td><strong>43. Fetch Module</strong></td>
<td>250</td>
</tr>
<tr>
<td><strong>Conventions</strong></td>
<td></td>
</tr>
<tr>
<td><code>fetch:binary</code></td>
<td>250</td>
</tr>
<tr>
<td><code>fetch:text</code></td>
<td>250</td>
</tr>
<tr>
<td><code>fetch:doc</code></td>
<td>251</td>
</tr>
<tr>
<td><code>fetch:binary-doc</code></td>
<td>251</td>
</tr>
<tr>
<td><code>fetch:content-type</code></td>
<td>252</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Changelog</strong></td>
<td></td>
</tr>
<tr>
<td><strong>44. File Module</strong></td>
<td>253</td>
</tr>
<tr>
<td><strong>Conventions</strong></td>
<td></td>
</tr>
<tr>
<td><code>File Paths</code></td>
<td>253</td>
</tr>
<tr>
<td><strong>Read Operations</strong></td>
<td></td>
</tr>
<tr>
<td><code>file:list</code></td>
<td>253</td>
</tr>
<tr>
<td><code>file:children</code></td>
<td>253</td>
</tr>
<tr>
<td><code>file:descendants</code></td>
<td>254</td>
</tr>
<tr>
<td><code>file:read-binary</code></td>
<td>254</td>
</tr>
<tr>
<td><code>file:read-text</code></td>
<td>254</td>
</tr>
<tr>
<td><code>file:read-text-lines</code></td>
<td>254</td>
</tr>
<tr>
<td><strong>Write Operations</strong></td>
<td></td>
</tr>
<tr>
<td><code>file:create-dir</code></td>
<td>255</td>
</tr>
<tr>
<td><code>file:create-temp-dir</code></td>
<td>255</td>
</tr>
<tr>
<td><code>file:create-temp-file</code></td>
<td>255</td>
</tr>
<tr>
<td><code>file:delete</code></td>
<td>255</td>
</tr>
<tr>
<td><code>file:write</code></td>
<td>256</td>
</tr>
<tr>
<td><code>file:write-binary</code></td>
<td>256</td>
</tr>
<tr>
<td><code>file:write-text</code></td>
<td>256</td>
</tr>
<tr>
<td><code>file:write-text-lines</code></td>
<td>256</td>
</tr>
<tr>
<td><code>file:append</code></td>
<td>257</td>
</tr>
<tr>
<td><code>file:append-binary</code></td>
<td>257</td>
</tr>
<tr>
<td><code>file:append-text</code></td>
<td>257</td>
</tr>
<tr>
<td><code>file:append-text-lines</code></td>
<td>257</td>
</tr>
<tr>
<td><code>file:copy</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:move</code></td>
<td>258</td>
</tr>
<tr>
<td><strong>File Properties</strong></td>
<td></td>
</tr>
<tr>
<td><code>file:exists</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:is-dir</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:is-absolute</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:is-file</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:last-modified</code></td>
<td>258</td>
</tr>
<tr>
<td><code>file:size</code></td>
<td>259</td>
</tr>
<tr>
<td><strong>Path Functions</strong></td>
<td></td>
</tr>
<tr>
<td><code>file:name</code></td>
<td>259</td>
</tr>
<tr>
<td><code>file:parent</code></td>
<td>259</td>
</tr>
<tr>
<td><code>file:path-to-native</code></td>
<td>259</td>
</tr>
<tr>
<td><code>file:resolve-path</code></td>
<td>259</td>
</tr>
<tr>
<td><code>file:path-to-uri</code></td>
<td>259</td>
</tr>
<tr>
<td><strong>System Properties</strong></td>
<td></td>
</tr>
<tr>
<td><code>file:dir-separator</code></td>
<td>260</td>
</tr>
<tr>
<td><code>file:path-separator</code></td>
<td>260</td>
</tr>
<tr>
<td><code>file:line-separator</code></td>
<td>260</td>
</tr>
</tbody>
</table>

xv

---

BaseX Documentation
file:temp-dir ........................................................................................................ 260
file:current-dir .................................................................................................. 260
file:base-dir ........................................................................................................ 260
Errors .................................................................................................................... 260
Changelog ............................................................................................................. 261
45. Full-Text Module ......................................................................................... 262
Conventions ........................................................................................................... 262
Database Functions .......................................................................................... 262
  ft:search ............................................................................................................ 262
  ft:tokens ............................................................................................................. 263
General Functions ............................................................................................. 263
  ft:contains ........................................................................................................ 263
  ft:count ............................................................................................................. 264
  ft:score .............................................................................................................. 264
  ft:_tokenize ................................................................................................. 264
  ft:normalize ................................................................................................. 265
  ft:thesaurus ................................................................................................. 265
Highlighting Functions ...................................................................................... 265
  ft:mark ............................................................................................................. 265
  ft:extract ........................................................................................................ 266
Errors .................................................................................................................... 266
Changelog ............................................................................................................. 266
46. Hashing Module .......................................................................................... 268
Conventions ........................................................................................................... 268
Functions ............................................................................................................. 268
  hash:md5 ......................................................................................................... 268
  hash:sha1 ......................................................................................................... 268
  hash:sha256 .................................................................................................... 268
  hash:hash ........................................................................................................ 268
Errors .................................................................................................................... 269
Changelog ............................................................................................................. 269
47. Higher-Order Functions Module ............................................................... 270
Conventions ........................................................................................................... 270
Loops ..................................................................................................................... 270
  hof:fold-left1 ............................................................................................... 270
  hof:until .......................................................................................................... 270
  hof:scan-left ............................................................................................... 271
  hof:take-while ............................................................................................. 271
  hof:drop-while ............................................................................................ 271
Sorting .................................................................................................................. 272
  hof:top-k-by ............................................................................................... 272
  hof:top-k-with .............................................................................................. 272
IDs ....................................................................................................................... 272
  hof:id .............................................................................................................. 272
  hof:const ....................................................................................................... 272
Changelog ............................................................................................................. 273
48. HTML Module ............................................................................................. 274
Conventions ........................................................................................................... 274
Functions ............................................................................................................. 274
  html:doc ......................................................................................................... 274
  html:parse ...................................................................................................... 274
  html:parser .................................................................................................... 274
Examples .............................................................................................................. 274
  Basic Example .............................................................................................. 274
  Specifying Options ....................................................................................... 275
  Parsing Binary Input .................................................................................... 275
Errors .................................................................................................................... 275
Changelog ............................................................................................................. 275
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>49. HTTP Client Module</td>
<td>276</td>
</tr>
<tr>
<td>Conventions</td>
<td>276</td>
</tr>
<tr>
<td>Functions</td>
<td>276</td>
</tr>
<tr>
<td>Examples</td>
<td>276</td>
</tr>
<tr>
<td>Status Only</td>
<td>276</td>
</tr>
<tr>
<td>Google Homepage</td>
<td>277</td>
</tr>
<tr>
<td>POST Request</td>
<td>278</td>
</tr>
<tr>
<td>File Upload</td>
<td>279</td>
</tr>
<tr>
<td>Errors</td>
<td>279</td>
</tr>
<tr>
<td>Changelog</td>
<td>280</td>
</tr>
<tr>
<td>50. Index Module</td>
<td>281</td>
</tr>
<tr>
<td>Conventions</td>
<td>281</td>
</tr>
<tr>
<td>Functions</td>
<td>281</td>
</tr>
<tr>
<td>index:facets</td>
<td>281</td>
</tr>
<tr>
<td>index:texts</td>
<td>281</td>
</tr>
<tr>
<td>index:attributes</td>
<td>281</td>
</tr>
<tr>
<td>index:tokens</td>
<td>282</td>
</tr>
<tr>
<td>index:element-names</td>
<td>282</td>
</tr>
<tr>
<td>index:attribute-names</td>
<td>282</td>
</tr>
<tr>
<td>Changelog</td>
<td>282</td>
</tr>
<tr>
<td>51. Inspection Module</td>
<td>283</td>
</tr>
<tr>
<td>Conventions</td>
<td>283</td>
</tr>
<tr>
<td>Reflection</td>
<td>283</td>
</tr>
<tr>
<td>inspect:functions</td>
<td>283</td>
</tr>
<tr>
<td>inspect:function-annotations</td>
<td>283</td>
</tr>
<tr>
<td>inspect:static-context</td>
<td>283</td>
</tr>
<tr>
<td>Documentation</td>
<td>284</td>
</tr>
<tr>
<td>inspect:types</td>
<td>284</td>
</tr>
<tr>
<td>inspect:function</td>
<td>285</td>
</tr>
<tr>
<td>inspect:context</td>
<td>285</td>
</tr>
<tr>
<td>inspect:module</td>
<td>286</td>
</tr>
<tr>
<td>inspect:xqdoc</td>
<td>286</td>
</tr>
<tr>
<td>Examples</td>
<td>286</td>
</tr>
<tr>
<td>Errors</td>
<td>288</td>
</tr>
<tr>
<td>Changelog</td>
<td>288</td>
</tr>
<tr>
<td>52. Job Module</td>
<td>289</td>
</tr>
<tr>
<td>Conventions</td>
<td>289</td>
</tr>
<tr>
<td>Services</td>
<td>289</td>
</tr>
<tr>
<td>Executing Jobs</td>
<td>289</td>
</tr>
<tr>
<td>job:eval</td>
<td>289</td>
</tr>
<tr>
<td>job:result</td>
<td>291</td>
</tr>
<tr>
<td>job:remove</td>
<td>292</td>
</tr>
<tr>
<td>job:wait</td>
<td>292</td>
</tr>
<tr>
<td>Listing Jobs</td>
<td>292</td>
</tr>
<tr>
<td>job:current</td>
<td>292</td>
</tr>
<tr>
<td>job:list</td>
<td>292</td>
</tr>
<tr>
<td>job:list-details</td>
<td>292</td>
</tr>
<tr>
<td>job:bindings</td>
<td>293</td>
</tr>
<tr>
<td>job:finished</td>
<td>293</td>
</tr>
<tr>
<td>job:services</td>
<td>293</td>
</tr>
<tr>
<td>Errors</td>
<td>293</td>
</tr>
<tr>
<td>Changelog</td>
<td>293</td>
</tr>
<tr>
<td>53. JSON Module</td>
<td>295</td>
</tr>
<tr>
<td>Conventions</td>
<td>295</td>
</tr>
<tr>
<td>Conversion Formats</td>
<td>295</td>
</tr>
<tr>
<td>Options</td>
<td>296</td>
</tr>
<tr>
<td>Functions</td>
<td>297</td>
</tr>
</tbody>
</table>
Functions

math:atan ................................................. 310
math:acos ................................................. 310
math:asin ................................................. 310
math:atan2 ............................................... 310
math:pow ................................................ 310
math:exp ................................................ 310
math:log ................................................ 310
math:log10 ............................................... 310

Additional Functions

math:e ..................................................... 311
math:sinh ............................................... 311
math:cosh ............................................... 311
math:tan ................................................ 311
math:asinh ............................................. 311
math:acosh ............................................. 311
math:atanh ............................................. 311
math:acosh ............................................. 311

Changelog ............................................... 311

57. Process Module ..................................... 312
Conventions ............................................. 312
Functions ............................................... 312
proc:system ........................................... 312
proc:execute .......................................... 312
## Profiling Module

- `prof:track` .................................................. 315
- `prof:time` .................................................. 316
- `prof:memory` ............................................... 316
- `prof:current-ms` ......................................... 316
- `prof:current-ns` ......................................... 316

## Debugging Functions

- `prof:dump` .................................................. 316
- `prof:variables` ........................................... 317
- `prof:type` .................................................. 317
- `prof:gc` .................................................... 317
- `prof:runtime` .............................................. 317

## Helper Functions

- `prof:void` .................................................. 317
- `prof:sleep` ................................................ 318
- `prof:human` .............................................. 318

## Errors

- .......................................................... 318

## Changelog

- .......................................................... 318

## Random Module

- .......................................................... 319

## Functions

- `random:double` .......................................... 319
- `random:integer` .......................................... 319
- `random:seeded-double` .................................. 319
- `random:seeded-integer` ................................ 319
- `random:gaussian` ........................................ 319
- `random:seeded-permutation` ......................... 320
- `random:uuid` ............................................ 320

## Errors

- .......................................................... 320

## Changelog

- .......................................................... 320

## Repository Module

- .......................................................... 321

## Functions

- `repo:install` ............................................ 321
- `repo:delete` ............................................ 321
- `repo:list` ................................................ 321

## Errors

- .......................................................... 321

## Changelog

- .......................................................... 321

## Request Module

- .......................................................... 323

## General Functions

- `request:method` ......................................... 323

## URI Functions

- `request:scheme` .......................................... 323
- `request:hostname` ....................................... 323
- `request:port` ............................................ 323
- `request:path` ............................................ 324
- `request:query` .......................................... 324
- `request:uri` .............................................. 324
- `request:context-path` .................................. 324

## Connection Functions

- .......................................................... 324
request:address .................................................................................................................. 324
request:remote-hostname ................................................................................................. 324
request:remote-address ................................................................................................. 324
request:remote-port ........................................................................................................ 324
Parameter Functions ........................................................................................................ 325
request:parameter-names ............................................................................................... 325
request:parameter ............................................................................................................ 325
Header Functions ............................................................................................................. 325
request:header-names ...................................................................................................... 325
request:header .................................................................................................................. 325
Cookie Functions ............................................................................................................ 325
request:cookie-names ....................................................................................................... 325
request:cookie .................................................................................................................. 326
Attribute Functions ......................................................................................................... 326
request:attribute-names .................................................................................................. 326
request:attribute .............................................................................................................. 326
request:set-attribute ....................................................................................................... 326
Errors .................................................................................................................................. 326
Changelog .......................................................................................................................... 326
62. RESTXQ Module ......................................................................................................... 328
Conventions ....................................................................................................................... 328
rest:base-uri .................................................................................................................... 328
rest:uri ............................................................................................................................... 328
rest:wadl ............................................................................................................................ 328
rest:init .............................................................................................................................. 328
Changelog .......................................................................................................................... 328
63. Session Module .......................................................................................................... 330
Conventions ....................................................................................................................... 330
Functions ............................................................................................................................. 330
session:id .......................................................................................................................... 330
session:created ............................................................................................................... 330
session:accessed ............................................................................................................... 330
session:names .................................................................................................................. 330
session:get ........................................................................................................................ 331
session:set ....................................................................................................................... 331
session:delete .................................................................................................................... 331
session:close ..................................................................................................................... 331
Errors .................................................................................................................................. 331
Changelog .......................................................................................................................... 331
64. Sessions Module ......................................................................................................... 333
Conventions ....................................................................................................................... 333
Functions ............................................................................................................................. 333
sessions:ids ........................................................................................................................ 333
sessions:created ............................................................................................................... 333
sessions:accessed ............................................................................................................... 333
sessions:names .................................................................................................................. 333
sessions:get ........................................................................................................................ 333
sessions:set ....................................................................................................................... 334
sessions:delete .................................................................................................................... 334
sessions:close .................................................................................................................... 334
Errors .................................................................................................................................. 334
Changelog .......................................................................................................................... 334
65. SQL Module .................................................................................................................. 335
Conventions ....................................................................................................................... 335
Functions ............................................................................................................................. 335
sql:init ............................................................................................................................... 335
sql:connect ....................................................................................................................... 335
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>66. Store Module</td>
<td>339</td>
</tr>
<tr>
<td>67. String Module</td>
<td>342</td>
</tr>
<tr>
<td>68. Unit Module</td>
<td>344</td>
</tr>
</tbody>
</table>

### Key-value operations
- store:get                      | 339  |
- store:put                      | 339  |
- store:get-or-put               | 339  |
- store:remove                   | 339  |
- store:keys                     | 340  |
- store:clear                    | 340  |

### Store Operations
- store:read                     | 340  |
- store:write                    | 340  |
- store:list                     | 340  |
- store:delete                   | 340  |

### Example
- unit:test                       | 344  |
- unit:before                     | 344  |
- unit:after                      | 345  |
- unit:before-module              | 345  |
- unit:after-module               | 345  |
- unit:ignore                     | 345  |

### Functions
- unit:assert                     | 345  |
- unit:assert-equals              | 345  |
- unit:fail                       | 346  |

### Example
- Unit Test Example               | 346  |
74. XQuery Module .......................................................... 370
    Conventions .......................................................... 370
    Evaluation ............................................................ 370
        xquery:eval ...................................................... 370
        xquery:eval-update ........................................... 371
    Parsing ............................................................... 371
        xquery:parse .................................................... 371
    Parallelized Execution ............................................. 372
        xquery:fork-join ............................................... 372
    Errors ............................................................... 372
    Changelog ........................................................... 373

75. XSLT Module .......................................................... 375
    Conventions .......................................................... 375
    Functions ............................................................ 375
        xslt:processor .................................................. 375
        xslt:version .................................................... 375
        xslt:transform .................................................. 375
        xslt:transform-text ............................................ 376
        xslt:transform-report ......................................... 376
    Examples ............................................................. 376
    Errors ............................................................... 378
    Changelog ........................................................... 378

VIII. Developing .......................................................... 379

76. Developing .......................................................... 380

77. Developing with Eclipse .............................................. 381
    Prerequisites ........................................................ 381
        Check Out ....................................................... 381
        Start in Eclipse ............................................... 381
        Alternative ..................................................... 382

78. Git ................................................................. 383
    Using Git to contribute to BaseX ................................ 383
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX. Web Technology</td>
<td>397</td>
</tr>
<tr>
<td>79. Maven</td>
<td>392</td>
</tr>
<tr>
<td>Using Maven</td>
<td>392</td>
</tr>
<tr>
<td>Artifacts</td>
<td>392</td>
</tr>
<tr>
<td>80. Releases</td>
<td>394</td>
</tr>
<tr>
<td>Official Releases</td>
<td>394</td>
</tr>
<tr>
<td>Stable Snapshots</td>
<td>394</td>
</tr>
<tr>
<td>Code Base</td>
<td>394</td>
</tr>
<tr>
<td>Maven Artifacts</td>
<td>394</td>
</tr>
<tr>
<td>Linux</td>
<td>394</td>
</tr>
<tr>
<td>IX. Web Technology</td>
<td>397</td>
</tr>
<tr>
<td>81. Translations</td>
<td>395</td>
</tr>
<tr>
<td>Working with the sources</td>
<td>395</td>
</tr>
<tr>
<td>Updating BaseX.jar</td>
<td>395</td>
</tr>
<tr>
<td>82. RESTXQ</td>
<td>398</td>
</tr>
<tr>
<td>Introduction</td>
<td>398</td>
</tr>
<tr>
<td>Preliminaries</td>
<td>398</td>
</tr>
<tr>
<td>Examples</td>
<td>399</td>
</tr>
<tr>
<td>Request</td>
<td>399</td>
</tr>
<tr>
<td>Constraints</td>
<td>399</td>
</tr>
<tr>
<td>Content Types</td>
<td>401</td>
</tr>
<tr>
<td>Parameters</td>
<td>402</td>
</tr>
<tr>
<td>Query Execution</td>
<td>403</td>
</tr>
<tr>
<td>Response</td>
<td>404</td>
</tr>
<tr>
<td>Custom Response</td>
<td>404</td>
</tr>
<tr>
<td>Forwards and Redirects</td>
<td>405</td>
</tr>
<tr>
<td>Output</td>
<td>405</td>
</tr>
<tr>
<td>Error Handling</td>
<td>406</td>
</tr>
<tr>
<td>Raise Errors</td>
<td>406</td>
</tr>
<tr>
<td>Catch XQuery Errors</td>
<td>407</td>
</tr>
<tr>
<td>Catch HTTP Errors</td>
<td>407</td>
</tr>
<tr>
<td>User Authentication</td>
<td>408</td>
</tr>
<tr>
<td>Functions</td>
<td>408</td>
</tr>
<tr>
<td>References</td>
<td>408</td>
</tr>
<tr>
<td>Changelog</td>
<td>408</td>
</tr>
<tr>
<td>83. Permissions</td>
<td>410</td>
</tr>
<tr>
<td>Preliminaries</td>
<td>410</td>
</tr>
<tr>
<td>Annotations</td>
<td>410</td>
</tr>
<tr>
<td>Permission Strings</td>
<td>410</td>
</tr>
<tr>
<td>Checking Permissions</td>
<td>411</td>
</tr>
<tr>
<td>Authentication</td>
<td>412</td>
</tr>
<tr>
<td>Changelog</td>
<td>412</td>
</tr>
<tr>
<td>84. WebSockets</td>
<td>414</td>
</tr>
<tr>
<td>Introduction</td>
<td>414</td>
</tr>
<tr>
<td>Protocol</td>
<td>414</td>
</tr>
<tr>
<td>Preliminaries</td>
<td>414</td>
</tr>
<tr>
<td>Configuration</td>
<td>414</td>
</tr>
<tr>
<td>Annotations</td>
<td>415</td>
</tr>
<tr>
<td>ws:connect(path)</td>
<td>415</td>
</tr>
<tr>
<td>ws:message(path, message)</td>
<td>415</td>
</tr>
<tr>
<td>ws:error(path, message)</td>
<td>415</td>
</tr>
<tr>
<td>ws:close(path)</td>
<td>415</td>
</tr>
<tr>
<td>ws:header-param(name, variable[, default])</td>
<td>415</td>
</tr>
<tr>
<td>Writing Applications</td>
<td>416</td>
</tr>
<tr>
<td>Examples</td>
<td>416</td>
</tr>
<tr>
<td>Basic Example</td>
<td>416</td>
</tr>
</tbody>
</table>
XI. Extensions .................................................................................................................. 456
  98. YAJSW ...................................................................................................................... 457
     Some basics of YAJSW .......................................................................................... 457
         Gather the files ............................................................................................... 457
         Install BaseX as a Windows Service .......................................................... 457
  99. Android ..................................................................................................................... 459
     Creating the Android Library Project ............................................................. 459
     Adjusting the Code ......................................................................................... 460
     Using the BaseX Android Library ................................................................... 463
XII. Advanced User's Guide ............................................................................................. 464
  100. Advanced User's Guide ....................................................................................... 465
    101. Catalog Resolver ............................................................................................... 466
         Introduction ..................................................................................................... 466
         Usage ............................................................................................................... 466
         Changelog ....................................................................................................... 468
    102. Storage Layout ................................................................................................... 469
         Data Types ....................................................................................................... 469
         Database Files .................................................................................................. 469
         Metadata, Name/Path/Doc Indexes: inf ....................................................... 469
         Node Table: tbl, tbli ....................................................................................... 470
         Texts: txt, atv ................................................................................................... 470
         Value Indexes: txtl, txtvr, atvl, atvr ........................................................... 470
         Document Path Index: pth ............................................................................ 470
         ID/Pre Mapping: idp ........................................................................................ 470
         Full-Text Fuzzy Index: ftxx, ftxy, ftxz ............................................................ 470
    103. Node Storage ....................................................................................................... 471
         Node Table ....................................................................................................... 471
         PRE Value ......................................................................................................... 471
         ID Value ........................................................................................................... 471
         Block Storage .................................................................................................. 471
    104. Binary Data .......................................................................................................... 473
         Storage ............................................................................................................. 473
         Usage ............................................................................................................... 473
    105. Parsers ................................................................................................................ 474
         XML Parsers ................................................................................................... 474
            GUI ................................................................................................................ 474
            Command Line ........................................................................................... 474
            XQuery ....................................................................................................... 474
         HTML Parser .................................................................................................... 474
            Installation .................................................................................................... 474
            Options ......................................................................................................... 475
         JSON Parser ...................................................................................................... 476
            GUI ................................................................................................................ 476
            Command Line ........................................................................................... 476
            XQuery ....................................................................................................... 476
         CSV Parser ....................................................................................................... 476
            GUI ................................................................................................................ 476
            Command Line ........................................................................................... 476
            XQuery ....................................................................................................... 476
         Text Parser ....................................................................................................... 477
            GUI ................................................................................................................ 477
            Command Line ........................................................................................... 477
            XQuery ....................................................................................................... 477
         Changelog ........................................................................................................... 477
    106. User Management ............................................................................................... 478
         Rules .................................................................................................................. 478
         Operations ....................................................................................................... 478
         Commands ...................................................................................................... 478

xxv
Part I. Main Page
Chapter 1. Main Page

Read this entry online in the BaseX Wiki.

BaseX, Graphical User Interface BaseX is a light-weight, high-performance and scalable XML Database and an XQuery 3.1 Processor with full support for the W3C Update and Full-Text extensions. It allows you to store, query and process large corpora of textual (XML, HTML, JSON, CSV, others) and binary data. The GUI provides an XQuery editor for writing complex applications and provides various visualizations to interactively explore data. RESTXQ enables Web Application development in XQuery. BaseX is platform-independent BSD-licensed (find more on Wikipedia).

This is the documentation for BaseX 10. The newest and upcoming features are highlighted.

If you have questions, or if you want to get into direct contact with the developer team and users of BaseX, please write to our mailing lists. Many questions are being discussed at StackOverflow; confirmed bugs and feature requests are discussed on GitHub.

Getting Started

The getting started section gives you a quick introduction to BaseX. We suggest that you 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.

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.

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 project.
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.
Chapter 2. BaseX 10

Read this entry online in the BaseX Wiki.

After 15 years of continuous development, the first double-digit version of BaseX is about to see the light of day soon.

We have taken the version jump as an opportunity to perform some major refactorings of BaseX, both under the hood and on API and XQuery level. Before migrating your projects to the new version, some adjustments may be required, so please read this article carefully.

Prerequisites

BaseX 10 requires Java 11 or later to run. Databases created with the new version are backward compatible and can still be opened with BaseX 9.

Migrating Applications

The following modifications might be relevant when migrating existing applications:

• The default ports for web applications have been changed from 8984/8985 to 8080/8081.

• The default admin password has been removed. The admin user can only be used if a password has been assigned, e.g., via the PASSWORD command.

• The conventions for functions in Clients in other programming languages were revised.

• The IGNOREHOSTNAME option was dropped and merged with IGNORECERT.

Storage

Whitespaces

All whitespaces are now preserved when importing XML resources, unless whitespace stripping is enabled.

The notorious CHOP option was removed to prevent conflicting behavior caused by earlier installations. It was replaced by a new STRIPWS option, which defaults to false. In addition, the new default of the serialization parameter indent is no.

Please be warned that the new default can throw off existing applications. If you want to restore the old behavior, you should assign the following values in your .basex configuration file, or the web.xml file of your Web Application:

| STRIPWS: true |
| SERIALIZER: indent=yes |

In the GUI editor, a shortcut and an icon were added to switch result indentation on and off.

In addition, databases may considerably increase in size, as whitespaces used for indenting an XML document will be interpreted and stored as additional text nodes. If your XML resources are structured and have no mixed content, it is advisable to enable whitespaces stripping when importing them to a database.

Value Resources

In addition to XML and binary resources, a third resource type has been added: XQuery values (atomic items and nodes, sequences, maps, arrays) can now be stored in databases as well. The db:put-value and db:get-value can be used to store to and retrieve values.

The new feature can e.g. be used to store maps in a database:
db:put-value(
  'factbook',
  map:merge(
    for $country in db:get('factbook')//country
      return map:entry($country/@name, $country//city/name ! string())
  ),
  'cities'
)

…and use them as index later on:

let $cities := db:get-value('factbook', 'cities')
for $country in ('Japan', 'Indonesia', 'Malaysia')
return $country || ': ' || string-join($cities?$country, ', ')

Backups

The Backup Commands and Backup Functions were enhanced to back up general data: registered users, scheduled services, key-value stores.

XQuery

Compilation

The compilation has been split up into multiple steps to improve locking.

So far, several internal steps were already performed when executing a query (see XQuery Optimizations for more details):

1. The query is parsed, i.e., the original query string is transformed to an executable tree representation.

2. External values that are passed on by APIs are bound to variables and the query context. External values can be names of databases, or contribute to a name that will later on be constructed in the query.

3. The query is compiled and evaluated.

The transaction manager gathers the names of the databases that will be accessed by a query. If it is not possible to uniquely identify all databases that may be opened by the query, global locking will be applied, and all databases will be locked. Detection can fail if the names of databases depend on external input. It can also fail if a query is too complex to associate character strings with database operations.

The compilation phase now comprises two separate steps:

1. Compilation of logical, context-independent (static) operations. External values are bound to the query, and deterministic code is rewritten, simplified and pre-evaluated.

2. Optimization of physical, context-based (dynamic) operations. Databases are opened and checked for available indexes; current date/time is retrieved. The resulting code is further rewritten and optimized.

Lock detection will be performed after the first step, and the code resulting from this step offers much more insight into which specific databases need to be locked. As a result, local locks can be applied to many more queries than before, and many queries can now run in parallel. An example:

```
declare variable $n external;
db:get('names-' || $n)
```

After the query has been parsed, a user-specific value (e.g., 123) will be bound to $n. The variable will be inlined by the compiler, and the argument of db:get will be pre-evaluated to names123. It is then easy for the lock detector to collect the name of the database that needs to read-locked before the query is eventually executed.

Another positive side effect of two-step compilation is that productive environments get faster in general: Queries can be compiled in parallel, and it’s only the optimization and evaluation of a query that may need to be delayed by locking.
Main-Memory Updates

XQuery Update provides constructs to **update XML nodes in main memory**. The data structures for in-memory representations of XML resources have been revised, such that updates can be performed orders of magnitudes faster than before. With BaseX 9.x, the following query runs for several minutes, whereas it can now be computed in a few seconds:

```xml
<xml>
  (1 to 1000000) ! <child/>
</xml> update {
  for $child at $pos in child
  return insert node text { $pos } into $child
}
```

Key-Value Store

A new **Store Module** provides functions to organize values in a persistent main-memory key-value store. The store allows you to speed up access to frequently accessed data.

Store data

```xml
let $email := map:merge(
  for $address in db:get('addressbook')//address
    return map:entry($address/name, $address/email)
) return store:put('emails', $email)
```

Retrieve data

```xml
let $name := 'Richard David James'
return store:get('email')($name)
```

The store is persistent: Its contents are written to disk if BaseX is shut down, and retrieved again after a restart.

Modules

Functions of all modules, excluding the **File Module**, now consistently resolve relative URI references against the static base URI, and not the current working directory.

Various modules and functions have been revised, added, renamed or removed:

<table>
<thead>
<tr>
<th>Description</th>
<th>BaseX 10</th>
<th>BaseX 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve XML resources</td>
<td><code>db:get</code></td>
<td><code>db:open</code></td>
</tr>
<tr>
<td>Retrieve nodes with specified pre values</td>
<td><code>db:get-pre</code></td>
<td><code>db:open-pre</code></td>
</tr>
<tr>
<td>Retrieve nodes with specified IDs</td>
<td><code>db:get-id</code></td>
<td><code>db:open-id</code></td>
</tr>
<tr>
<td>Retrieve binary resources</td>
<td><code>db:get-binary</code></td>
<td><code>db:retrieve</code></td>
</tr>
<tr>
<td>Retrieve value resources</td>
<td><code>db:get-value</code></td>
<td>new</td>
</tr>
<tr>
<td>Add or replace resource</td>
<td><code>db:put</code>, arguments swapped!</td>
<td><code>db:replace</code></td>
</tr>
<tr>
<td>Add or replace binary resource</td>
<td><code>db:put-binary</code>, arguments swapped!</td>
<td><code>db:store</code></td>
</tr>
<tr>
<td>Add or replace value resource</td>
<td><code>db:put-value</code></td>
<td>new</td>
</tr>
<tr>
<td>Get resource type</td>
<td><code>db:type</code></td>
<td><code>db:is-raw, db:is-xml</code></td>
</tr>
<tr>
<td>Fetch XML document</td>
<td><code>fetch:doc</code></td>
<td><code>fetch:xml</code></td>
</tr>
<tr>
<td>Convert binary data to XML</td>
<td><code>fetch:binary-doc</code></td>
<td><code>fetch:xml-binary</code></td>
</tr>
<tr>
<td>Module: Process Geo data</td>
<td>removed</td>
<td>Geo Module</td>
</tr>
</tbody>
</table>
XQuery jobs

<table>
<thead>
<tr>
<th>Description</th>
<th>BaseX 10</th>
<th>BaseX 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return variable bindings of a job</td>
<td>job:bindings</td>
<td>new</td>
</tr>
<tr>
<td>Return variable bindings of a job</td>
<td>job:remove</td>
<td>jobs:stop</td>
</tr>
<tr>
<td>Module: Main-memory key-value store</td>
<td>Store Module</td>
<td>new</td>
</tr>
<tr>
<td>Module: String computations</td>
<td>String Module</td>
<td>Strings Module</td>
</tr>
<tr>
<td>Format string</td>
<td>string:format</td>
<td>out:format</td>
</tr>
<tr>
<td>Return control characters</td>
<td>string:cr,</td>
<td>out:cr,</td>
</tr>
<tr>
<td></td>
<td>string:tab</td>
<td>out:nl,</td>
</tr>
<tr>
<td>Module: Process ZIP files</td>
<td>removed</td>
<td>ZIP Module</td>
</tr>
</tbody>
</table>

## Commands

The following commands have been revised:

<table>
<thead>
<tr>
<th>Description</th>
<th>BaseX 10</th>
<th>BaseX 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>List directories and resources.</td>
<td>DIR</td>
<td>new</td>
</tr>
<tr>
<td>Retrieve single XML document</td>
<td>GET</td>
<td>new</td>
</tr>
<tr>
<td>Retrieve binary resource</td>
<td>BINARY GET</td>
<td>RETRIEVE</td>
</tr>
<tr>
<td>Add or replace resources</td>
<td>PUT</td>
<td>REPLACE</td>
</tr>
<tr>
<td>Store binary resource</td>
<td>BINARY PUT</td>
<td>STORE</td>
</tr>
<tr>
<td>Returns current option values</td>
<td>SHOW OPTIONS</td>
<td>GET</td>
</tr>
<tr>
<td>Lists jobs</td>
<td>removed</td>
<td>JOBS LIST</td>
</tr>
<tr>
<td>Returns a job result</td>
<td>removed</td>
<td>JOBS RESULT</td>
</tr>
<tr>
<td>Stops a job</td>
<td>removed</td>
<td>JOBS STOP</td>
</tr>
</tbody>
</table>

## HTTP Requests

HTTP requests in BaseX take advantage of the new Java HTTP Client. This client provides a better overall performance, uses internal connection pools and follows redirects across different protocols (http, https).

HTTP operations are, among others, performed by:

- the HTTP Client Module;
- the Fetch Module, Database Module, Fetch Module, Validation Module, XSLT Module or Repository Module;
- fn:doc and fn:collection;
- the CREATE DB and REPO INSTALL commands.

## Catalogs

From early on, catalog resolvers had been neglected both in BaseX and Java. This has changed: The new XML Catalog API from Java is universally used to resolve references to external resources. As an alternative, Norman Walsh’s Enhanced XML Resolver is utilized if it is found in the classpath.

The option for supplying the XML catalog was renamed from CATFILE to CATALOG. See Catalog Resolver for more details.

## Graphical User Interface

The graphical user interface of BaseX has been revised and made more consistent.
The icons were replaced by scalable ones, building upon the HiDPI graphics support for Windows and Linux.

**REST**

Results in the `rest` namespace are now returned without prefix:

```xml
<!-- before -->
<rest:databases xmlns:rest="http://basex.org/rest"/>

<!-- now -->
<databases xmlns="http://basex.org/rest"/>
```

When listing the resources of a database, `dir` elements are returned for resources that are located in subdirectories. See **REST** for more details.
Part II. Getting Started
Chapter 3. Getting Started

Read this entry online in the BaseX Wiki.

This page is one of the Main Sections of the documentation. It gives you a brief introduction on how to start, run, and use BaseX. After you have set up BaseX, we suggest that you should start with the Graphical User Interface.

Overview

First Steps

- **Startup**: How to get BaseX running
- **Graphical User Interface** (see available Shortcuts)
- **Command-Line Client**: Use BaseX in your bash
- **Database Server**: The client/server architecture
- **Web Application**: The HTTP server
- **DBA**: Browser-based database administration

Command Line

- **Command-Line Options**
- **Start Scripts**

General Info

- **Databases**: How databases are created, populated and deleted
- **Commands**: Full overview of all database commands
- **Options**: Listing of all database options
- **Configuration**: BaseX start files and directories

Editing XML and XQuery Files

We encourage you to use the BaseX Editor to run your queries and edit your XML data.

- **Integrating oXygen**
- **Integrating Eclipse**
- **Integrating IntelliJ IDEA**

Tutorials and Slides

**BaseX: Introduction**

- Tamara Marnell: BaseX for Newbies
- Paul Swennenhuis: BaseX for Dummies: Part I, Part I (files), Part II
- Neven Jovanović: BaseX Adventures
• Imed Bouchrika: Tutorial. Using a native XMLDBS

• Farid Djaïdja: XQuery pour les Humanités Numériques (French)

**XML and XQuery**

• XML Technologies. University course on XML, XPath, XQuery, XSLT, Validation, Databases, etc.

• XQuery: A Guided Tour. From the book "XQuery from the Experts".

• XQuery Summer Institute. Exercises and Answers.

• W3 Schools XQuery Tutorial. Not affiliated with W3C.

**BaseX: Talks, Questions**

• Our Mailing List. Join and contribute.

• Stack Overflow. Questions on baseX.

• GitHub Issue Tracker. Confirmed bugs and feature requests.

• XML Prague User Meetings. Slides and videos.
Chapter 4. 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.

Introduction

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

1. BaseX comes with a Graphical User Interface that offers you tools for managing, querying and visualizing your data and writing sophisticated applications in XQuery.

2. You can start BaseX as a standalone Command-Line Client if you prefer to work in the terminal, or want to do batch processing.

3. The Database Server is the right choice if you have multiple users or clients, or if you use other programming languages and if you don’t require HTTP services.

4. The HTTP Server provides REST and WebDAV services. With RESTXQ, complex web applications can be built, and the embedded DBA application allows you to work with BaseX in the browser.

5. It can also be embedded as a Java library in your applications.

BaseX has been tested on numerous platforms, including Windows (2000, XP, Vista, 7, 10, 11), Mac OS X (10.x, 11.x), Linux (SuSE xxx, Debian, Redhat, CentOS, Ubuntu) and OpenBSD (up to 7.x). It is platform-independent and runs on any system supporting Java.

Startup

First, ensure that an up-to-date version of the Java Runtime Environment (JRE) is available:

<table>
<thead>
<tr>
<th>BaseX Version</th>
<th>Minimum Java Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>older</td>
<td>6</td>
</tr>
</tbody>
</table>

If you have several versions or distributions installed, you can type `java -version` on command-line to check which Java version is currently used.

If you have Windows, we recommend the .msi distributions from Adoptium. The JRE packages are sufficient, but you can also install the JDK (Java Development Kit).

Next, get a fresh copy of BaseX from our homepage. The following distributions are available:

Core Package

The Core Package is a very compact JAR file. It contains the BaseX database management system, the XQuery processor, the client/server architecture, and the graphical user interface. It runs without additional libraries.

Full Distributions

In addition, the ZIP Package and the Windows Installer contain extra libraries for RESTXQ web applications and other advanced features, Start Scripts, and the DBA, a browser-based database administration interface. If you unzip or install BaseX, the target directory will contain the following directories:
### Startup

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>Start scripts (Windows, Linux).</td>
</tr>
<tr>
<td>data</td>
<td>The database directory.</td>
</tr>
<tr>
<td>etc</td>
<td>Example data: XML sample, catalog and DTD files.</td>
</tr>
<tr>
<td>lib</td>
<td>Extra libraries (Jetty, Tagsoup, …).</td>
</tr>
<tr>
<td>lib/custom</td>
<td>Directory in which additional JAR files can be placed (such as the Saxon library).</td>
</tr>
<tr>
<td>repo</td>
<td>Repository for external XQuery modules (the FunctX library is included as example).</td>
</tr>
<tr>
<td>src</td>
<td>Directory for your XQuery scripts and other source data.</td>
</tr>
<tr>
<td>webapp</td>
<td>Web Application directory: home of the RESTXQ web application, REST scripts, and DBA.</td>
</tr>
</tbody>
</table>

Global Options are stored in the `.baxex configuration file.`

If BaseX is started via the start scripts or the Windows icons, all JAR files in the `lib` directory and its descendant directories will be added to the class path.

If you work with the ZIP distribution, and if you want to make BaseX globally available, you can add the `bin` directory to your PATH environment variable.

### Web Archive

The **WAR Archive** can be embedded in existing Java web servers.

### Other Distributions

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

### Concurrent Operations

If you want to perform parallel (concurrent) read and write operations on your databases, you must use the client/server architecture or run BaseX as a web application. You can safely open a database in different JVMs (Java virtual machines) for read-only access, and you will not encounter any problems when reading from and writing to different databases. Update operations from different JVMs to the same database will be rejected or may even lead to corrupt databases.

For example, if you only read data, you can easily run several clients (standalone, GUI, database clients) in parallel. If you update your data, however, you shouldn’t use the GUI or a standalone instance at the same time.

More details on concurrency can be found on the **Transaction Management** page.

### Changelog

**Version 10.0**

- Update: Switched to Java 11

**Version 9.0**

- Update: Switched to Java 8

**Version 8.0**

- Update: Switched to Java 7
Version 7.0

• Updated: BaseXJAXRX has been replaced with BaseXHTTP
Chapter 5. Graphical User Interface

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. BaseX comes with a graphical user interface that offers you great tools for managing, querying and visualizing your data and write complex applications in XQuery.

Startup

The graphical user interface can be started as follows:

- If you have installed BaseX on Windows, click on the BaseX GUI icon.

- Run one of the basexgui or basexgui.bat scripts.

- You can also double-click on the BaseX.jar file (this way, no libraries will be added).

- For developers: type in mvn exec:java in the main directory of the baseX project.

Some additional command-line options are available.

It is worth mentioning that the standalone client must not be used if you perform parallel (concurrent) read and write operations on your databases. See Concurrent Operations for more details.

Introduction

At the top of the BaseX window, the menu bar resides. It houses all important features of the BaseX GUI. Use the Database menu to create and manage your XML databases. The Editor menu gives you access to a variety of tools and options for working with files. The View menu lets you toggle between the bars and panels described below. The Visualization menu offers you a comprehensive set of data representations that will help you to understand your data even better. Use the Options menu to set your preferences regarding real-time execution, colors, fonts, and packages.

Right below the menu bar, you can find the Buttons bar and just below that the Input Bar. The Buttons bar offers you a wide range of shortcuts, mostly for menu options, such as managing databases and displaying views and visualizations, but also for navigating through your data. With the Input Bar, you can query your data using three different kinds of query syntax.

The Status Bar is situated at the bottom of the BaseX window.

The BaseX editor consists of the Project view, a file browser with optional input fields for searching files, and the actual Editor panel with buttons for creating, opening, saving, searching, executing and debugging your files.
In addition to that, the Result view displays the output of queries and database operations and the Info view shows you information about database processes and query execution.

To gain further insights into your data, you can choose to display various visualizations such as Map, Tree, Folder, Plot, Table and Explorer.

Database Management

The BaseX GUI is a great place for creating and managing your XML databases.

To create a new database, select Database → New from the menu and browse to an XML document of your choice. You can start with the factbook.xml document, which contains statistical information on the world’s countries. It is included in the etc directory of our full distributions (ZIP Package and Windows Installer) or can be downloaded here (1.3 MB). In the Create Database dialog, specify the path to your input file and the name of the new database. If you leave the input file field empty, an empty database will be created. Click the OK button to create the database.

Note: You can also use the GUI’s editor to create and edit your own XML document. Just specify it as input file for the creation of a new database after saving the document to disk.

To open, rename, copy or drop a database, choose Database → Open & Manage... from the menu. Select one of the available databases on the left-hand side and click on one of the buttons on the right: Open, Rename, Copy or Drop. To open a database, you can also double-click on the database name.

Opening a database activates three more options in the Database menu:

• The Properties item gives you access to a variety of database options and information:
  • Add resources and/or set parsing preferences.
  • Gain insights into element and attribute names, paths and other meta information.
Graphical User Interface

• Create and manage text, attribute, token and full-text indexes. Customize indexes by specifying language, stemming, case-sensitivity and diacritics settings or include a stop word list.

• With the Export item, you can serialize your database into a whole range of different output formats, including XML, JSON and CSV.

• The Close item closes the database. An open database is closed automatically as soon as another database is opened.

Note: You can also access the menu options New, Open & Manage, Properties and Close from BaseX’s Buttons bar.

Editor

The built-in editor of BaseX is a powerful tool for write XQuery code and Command Scripts, editing text documents (XML, JSON, JavaScript, ..), and developing RESTXQ applications:

• The editor offers native syntax highlighting for XQuery, XML, JSON and JavaScript.

• XQuery, XML and JSON files will be parsed in real time and errors will be highlighted.

• XQuery code and command scripts can be executed (via Ctrl Enter or by clicking on the green triangle).

Numerous keyboard shortcuts are available to speed up editing and debugging. Some examples:

• Ctrl H: Search for the currently selected string in your complete project.

• Ctrl .: Jump to the next erroneous code in your project.

If you right-click on an XML document in the Project view, the selected file will be parsed and bound to the context item:
Project View

The Project view is attached to the Editor panel. It displays all files of the current project directory in a tree structure. Files can be renamed and deleted by right-clicking on the files. The project directory can be changed as well; the most recent directories will be kept in the history.

All XQuery files in the project directory will be parsed in the background. Buggy XQuery modules, and files importing these modules, will be marked red. With the text fields on top, you can interactively search for file names and contents.

If a directory contains a .ignore file, its files and contents will be ignored.

Input Bar

The Input Bar is situated on top of the main window. It offers you three different modes to query your XML databases: Find, XQuery and Command.

The upcoming example queries can all be used with an instance of the factbook database:

Find

In the Find mode, the input bar can be used to find single elements and texts in the currently opened database. The following syntax is supported:

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>city</td>
<td>Find elements named city, and texts containing this token.</td>
</tr>
<tr>
<td>=India</td>
<td>Find texts matching the exact string India.</td>
</tr>
<tr>
<td>~Cingdom</td>
<td>Find texts equal or similar to the token Cingdom.</td>
</tr>
<tr>
<td>@id</td>
<td>Find attributes named id and attribute values containing this token.</td>
</tr>
<tr>
<td>@=f0_119</td>
<td>Find attribute values matching the exact string f0_119.</td>
</tr>
<tr>
<td>&quot;European Chinese&quot;</td>
<td>Find texts containing the phrase &quot;European Chinese&quot;.</td>
</tr>
<tr>
<td>//city</td>
<td>Leading slash: Interpret the input as XPath expression (see below).</td>
</tr>
</tbody>
</table>

XQuery

In the XQuery mode, XPath and XQuery expressions can be entered in the input bar.

To evaluate the following example queries, type them in the input bar and press Enter or click on the Run query button (green triangle) adjacent to the input bar:

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>//country</td>
<td>Return all country elements.</td>
</tr>
<tr>
<td>//country[name = &quot;Switzerland&quot;]</td>
<td>Return the country element of &quot;Switzerland&quot;.</td>
</tr>
<tr>
<td>for $city in //city where $city/population &gt; 1000000 order by $city/ascending return $city/name</td>
<td>Return the names of all cities with a population larger than one million and order the results by the name of the city.</td>
</tr>
</tbody>
</table>

Command

In the Command mode, BaseX Commands can be entered and executed. Just try the following examples:
Graphical User Interface

- **INFO**: Returns system information.
- **CREATE DB TEST**: Creates an empty database named "TEST".
- **LIST**: Lists all databases.

Visualizations

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

**Result (View menu)**

Displays query results and other textual output (e.g., content of the currently open database). Query results can be saved in a file.

**Info (View menu)**

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

**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**

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**

**Plot**
Graphical User Interface

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

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

Explorer

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.

Real-time Options

In the Options menu, you can change how queries are executed and visualized:

• **Real-time Execution**: If real-time execution is enabled, your searches and queries will be executed with each key click and the results will be instantly shown.

• **Real-time Filtering**: If enabled, all visualizations will be limited to the actual results in real-time. If this feature is disabled, the query results are highlighted in the visualizations and can be explicitly filtered with the 'Filter' button.

Look and Feel

By default, the Look and Feel of your operating system will be used in the GUI. In the Preferences dialog, you can choose from additional 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 themes will get available.

Changelog

Version 9.3

• Updated: Project View: ignore directories with .ignore file

Version 9.1

• Added: Project View, set XML document as context.
Graphical User Interface

Version 8.4
• Added: highlighting of erroneous XQuery modules in the project view.

Version 8.0
• Updated: support for dark look and feels; support for JTattoo library
Chapter 6. Shortcuts

Read this entry online in the BaseX Wiki.

This article is about the GUI of BaseX. It gives you an overview of the most important hotkeys available in the visual frontend.

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.

Query Execution

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute XQuery</td>
<td>Ctrl Enter</td>
<td># Enter</td>
</tr>
<tr>
<td>Execute XQUnit tests</td>
<td>Ctrl Shift Enter</td>
<td># Shift Enter</td>
</tr>
<tr>
<td>Edit external variables</td>
<td>Ctrl Shift E</td>
<td># Shift E</td>
</tr>
<tr>
<td>Result indentation on/off</td>
<td>Ctrl Shift I (new)</td>
<td># Shift I (new)</td>
</tr>
</tbody>
</table>

Custom Editing

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs Code Completions</td>
<td>Ctrl Space</td>
<td>Ctrl Space</td>
</tr>
<tr>
<td>Sort lines</td>
<td>Ctrl U</td>
<td># U</td>
</tr>
<tr>
<td>(Un)comment selection/line</td>
<td>Ctrl K</td>
<td># K</td>
</tr>
<tr>
<td>Delete line(s)</td>
<td>Ctrl Shift D</td>
<td># Shift D</td>
</tr>
<tr>
<td>Duplicate line(s)</td>
<td>Ctrl D</td>
<td># D</td>
</tr>
<tr>
<td>Lower case</td>
<td>Ctrl Shift L</td>
<td># Shift L</td>
</tr>
<tr>
<td>Upper case</td>
<td>Ctrl Shift U</td>
<td># Shift U</td>
</tr>
<tr>
<td>Title case</td>
<td>Ctrl Shift T</td>
<td># Shift T</td>
</tr>
<tr>
<td>Format code (experimental)</td>
<td>Ctrl Shift F</td>
<td># Shift F</td>
</tr>
</tbody>
</table>

Finding

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search highlighted string in project</td>
<td>Ctrl H</td>
<td># Shift H</td>
</tr>
<tr>
<td>Jump to next error in project</td>
<td>Ctrl . (period)</td>
<td># . (period)</td>
</tr>
</tbody>
</table>
### Shortcuts

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jump to currently edited file</td>
<td>Ctrl J</td>
<td># J</td>
</tr>
<tr>
<td>Go to line</td>
<td>Ctrl L</td>
<td># L</td>
</tr>
<tr>
<td>Find and replace text</td>
<td>Ctrl F</td>
<td># F</td>
</tr>
<tr>
<td>Find next instance of text</td>
<td>F3</td>
<td># F3# G</td>
</tr>
<tr>
<td>Find previous instance of text</td>
<td>Shift F3</td>
<td>Ctrl Shift G</td>
</tr>
<tr>
<td>Jump to file history</td>
<td>Ctrl F6</td>
<td># F6 (new)</td>
</tr>
</tbody>
</table>

### Standard Editing

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo recent changes</td>
<td>Ctrl Z</td>
<td># Z</td>
</tr>
<tr>
<td>Redo recent changes</td>
<td>Ctrl Y</td>
<td># Shift Z</td>
</tr>
<tr>
<td>Cut selection</td>
<td>Ctrl X</td>
<td># X</td>
</tr>
<tr>
<td>Copy selection to clipboard</td>
<td>Ctrl C</td>
<td># C</td>
</tr>
<tr>
<td>Paste from clipboard</td>
<td>Ctrl V</td>
<td># V</td>
</tr>
<tr>
<td>Select All</td>
<td>Ctrl A</td>
<td># A</td>
</tr>
<tr>
<td>Delete character left of cursor</td>
<td>Backspace</td>
<td>Backspace</td>
</tr>
<tr>
<td>Delete character right of cursor</td>
<td>Delete</td>
<td>Delete (fn Backspace)</td>
</tr>
<tr>
<td>Delete word left of cursor</td>
<td>Ctrl Backspace</td>
<td>Alt Backspace</td>
</tr>
<tr>
<td>Delete word right of cursor</td>
<td>Ctrl Delete</td>
<td>Alt Delete</td>
</tr>
<tr>
<td>Delete text left of cursor</td>
<td>Ctrl Shift Backspace</td>
<td># Backspace</td>
</tr>
<tr>
<td>Delete text right of cursor</td>
<td>Ctrl Shift Delete</td>
<td># Delete</td>
</tr>
</tbody>
</table>

### Navigation

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move one character to the left/right</td>
<td>←/→</td>
<td>←/→</td>
</tr>
<tr>
<td>Move one word to the left/right</td>
<td>Ctrl ←/→</td>
<td>Alt ←/→</td>
</tr>
<tr>
<td>Move to beginning/end of line</td>
<td>Home/End</td>
<td># ←/→</td>
</tr>
<tr>
<td>Move one line up/down</td>
<td>↑/↓</td>
<td>↑/↓</td>
</tr>
<tr>
<td>Move one screen-full up/down</td>
<td>Page ↑/↓</td>
<td>Page ↑/↓ (fn ↑/↓)</td>
</tr>
<tr>
<td>Move to top/bottom</td>
<td>Ctrl Home/End</td>
<td>#/# (# ↑/↓)</td>
</tr>
<tr>
<td>Scroll one line up/down</td>
<td>Ctrl ↑/↓</td>
<td>Alt ↑/↓</td>
</tr>
</tbody>
</table>

### GUI

### Global Shortcuts

The following shortcuts are available from most GUI components:

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus input bar</td>
<td>F8 (before: F6)</td>
<td># F8 (before: # F6)</td>
</tr>
<tr>
<td>Focus editor view</td>
<td>F12</td>
<td># F12</td>
</tr>
<tr>
<td>Focus result view</td>
<td>Shift F12 (new)</td>
<td>Shift # F12 (new)</td>
</tr>
<tr>
<td>Jump to next/previous panel</td>
<td>Ctrl (Shift) Tab</td>
<td>Ctrl (Shift) Tab</td>
</tr>
<tr>
<td>Increase/Decrease font size</td>
<td>Ctrl +/-</td>
<td># +/-</td>
</tr>
</tbody>
</table>
### Shortcuts

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset font size</td>
<td>Ctrl 0</td>
<td># 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse back/forward</td>
<td>Alt ←/#Backspace</td>
<td># ←/→</td>
</tr>
<tr>
<td>Browse one level up</td>
<td>Alt ↑</td>
<td># ↑</td>
</tr>
<tr>
<td>Browse to the root node</td>
<td>Alt Home</td>
<td># Home</td>
</tr>
</tbody>
</table>

### Menu Shortcuts

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

#### Database

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new database</td>
<td>Ctrl N</td>
<td># N</td>
</tr>
<tr>
<td>Open/manage existing databases</td>
<td>Ctrl M</td>
<td># M</td>
</tr>
<tr>
<td>View/edit database properties</td>
<td>Ctrl D</td>
<td># D</td>
</tr>
<tr>
<td>Close opened database</td>
<td>Ctrl Shift W</td>
<td># Shift W</td>
</tr>
<tr>
<td>Exit application</td>
<td>Ctrl Q</td>
<td># Q</td>
</tr>
</tbody>
</table>

#### Editor

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new tab</td>
<td>Ctrl T</td>
<td># T</td>
</tr>
<tr>
<td>Open existing file</td>
<td>Ctrl O</td>
<td># O</td>
</tr>
<tr>
<td>Save file</td>
<td>Ctrl S</td>
<td># S</td>
</tr>
<tr>
<td>Save copy of file</td>
<td>Ctrl Shift S</td>
<td># Shift S</td>
</tr>
<tr>
<td>Close tab</td>
<td>Ctrl W, Ctrl F4</td>
<td># W, # F4</td>
</tr>
</tbody>
</table>

#### View

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle query/text editor</td>
<td>Ctrl E</td>
<td># E</td>
</tr>
<tr>
<td>Toggle project structure</td>
<td>Ctrl P</td>
<td># P</td>
</tr>
<tr>
<td>Toggle result view</td>
<td>Ctrl R</td>
<td># R</td>
</tr>
<tr>
<td>Toggle query info view</td>
<td>Ctrl I</td>
<td># I</td>
</tr>
</tbody>
</table>

#### Options

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open preference dialog</td>
<td>Ctrl Shift P</td>
<td># , (comma)</td>
</tr>
</tbody>
</table>

#### Visualization

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle map view</td>
<td>Ctrl 1</td>
<td># 1</td>
</tr>
<tr>
<td>Toggle tree view</td>
<td>Ctrl 2</td>
<td># 2</td>
</tr>
</tbody>
</table>
Shortcuts

<table>
<thead>
<tr>
<th>Feature</th>
<th>Ctrl Key</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toggle folder view</td>
<td>Ctrl 3</td>
<td>3</td>
</tr>
<tr>
<td>Toggle plot view</td>
<td>Ctrl 4</td>
<td>4</td>
</tr>
<tr>
<td>Toggle table view</td>
<td>Ctrl 5</td>
<td>5</td>
</tr>
<tr>
<td>Toggle explorer view</td>
<td>Ctrl 6</td>
<td>6</td>
</tr>
</tbody>
</table>

Help

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Help</td>
<td>F1</td>
<td>F1</td>
</tr>
</tbody>
</table>

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

Changelog

Version 10.0

- Updated: Some shortcuts were modified.

Version 8.4

- Added: Duplicate line (Ctrl D)

Version 8.4

- Added: Lower case (Ctrl Shift L), Upper case (Ctrl Shift U), Title case (Ctrl Shift T)

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 line(s) (Ctrl Shift D), jump to highlighted error (Ctrl .)
Chapter 7. Command-Line Client

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. It introduces you to the standalone command-line mode of BaseX.

Startup

The command-line client can be started as follows:

- Run one of the basex or basex.bat scripts.
- If you have installed BaseX on Windows, click on the BaseX Standalone icon.

All operations will be performed with admin permissions (no password needs to be supplied). Various command-line options are available to simplify batch processing. The start script can be adjusted for individual purposes (e.g. if the default memory limit is too restrictive).

The standalone client must not be used if you perform parallel (concurrent) read and write operations on your databases. See Concurrent Operations for more details.

Operations

Create a Database

To create a database, you need an XML document, e.g., factbook.xml. Save this document to your working directory and type in the following command to create and open the database:

```
> CREATE DB factbook

factbook is the name of the database

factbook.xml is the initial input of the database
```

By default, databases are stored in the basex/data directory of your project’s home directory. Depending on your Configuration, the location may vary.

Execute a Query

The Template:Commands command lets you run a query. The following query returns all country elements of the currently opened database:

```
> XQUERY //country
```

You can also run queries in files:

```
> RUN /path/to/query.xq
```

Database Commands

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

```
> LIST
```

To open an existing database, execute the following:

```
> OPEN factbook
```

To get information on the currently opened database, type:

```
> INFO
```
You can also address a database within your query with the `db:get` function:

```plaintext
> XQUERY db:get("factbook")//country
```

To close the current database, please type:

```plaintext
> CLOSE
```

A database can eventually be dropped again:

```plaintext
> DROP DB factbook
```

**Multiple Resources**

One database can contain not only a single, but millions of documents. All documents can have a different structure.

With the following commands, you can create an empty database and add two documents. It is also possible to address resources via URLs:

```plaintext
> CREATE DB store , > ADD factbook.xml , > ADD http://files.basex.org/xml/xmark.xml
```

Deleting a document from a database is easy, but make sure that the database, which contains the addressed document, is currently opened:

```plaintext
> DELETE factbook.xml
```

**Backup and Restore**

To back up and restore your database, type:

```plaintext
> CREATE BACKUP factbook , > RESTORE factbook
```

The backup file is stored in the database directory. It contains the name of the database and a timestamp: `[db-name]-[timestamp].zip`. If a database is to be restored, and if several backups exist, the backup with the newest timestamp is taken.
Chapter 8. Database Server

Read this entry online in the BaseX Wiki.

This article belongs to the Getting Started Guide. It tells you how to run BaseX in client-server mode from command-line.

Startup

Server

*With Version 10, the default admin password has been removed.*

The database server handles concurrent read and write transactions, manages user permissions and logs user interactions. It can be started as follows:

- Run one of the `basexserver` or `basexserver.bat` scripts. Use `basexserverstop` or `basexserverstop.bat` to gracefully shut down the server.

- If you have installed BaseX on Windows, click on the BaseX HTTP Server (Start) icon, which will start both the HTTP Server used for Web Applications and the database server. With BaseX HTTP Server (Stop), you can shut down the server process.

Unless you have already chosen an admin password yet (e.g., via the Windows installer or a previous installation), you can do so by invoking the `PASSWORD` command on your terminal:

```
basexserver -c PASSWORD
BaseX [Server]
Server was started (port: 1984).
Password: _
```

By default, the server listens to the port 1984. Pressing Ctrl+c will close all connections and databases and gracefully shut down the server process.

Various command-line options are available to simplify batch processing. The start script can be adjusted for individual purposes (e.g. if the default memory limit is too restrictive).

Client

Database clients are started similarly:

- Run one of the `basexclient` or `basexclient.bat` scripts.

- Execute the following command: `java -cp BaseX.jar org.basex.BaseXClient`

- If you have installed BaseX on Windows, click on the BaseX Client icon.

At startup, you need to enter your credentials.

For further details, have a look at the command-line options and the start script.

Introduction

The BaseX command-line client provides similar features to the standalone client. The major difference is that all commands will be executed by the BaseX server instance. As a consequence, paths/URIs to resources need to be resolvable by the server (file contents will not be transferred to the server).

Username and password can also be specified as command-line option. To evaluate commands without entering the console mode, you can use the `-c` option on the command line:
BaseXClient -V -U admin -P... -c "CREATE DB input <example/>; XQUERY /

Database 'input' created in 13.85 ms.
Query:
/

Parsing: 0.18 ms
Compiling: 0.04 ms
Evaluating: 0.12 ms
Printing: 0.07 ms
Total Time: 0.41 ms

Hit(s): 1 Item
Updated: 0 Items
Printed: 10 Bytes
Read Locking: local [input]
Write Locking: none

Query "user" executed in 0.41 ms.

Language Bindings

If you want to communicate with the database server programmatically, we provide clients for various programming languages.

Changelog

Version 10.0

- Updated: The default admin password has been removed.
Chapter 9. Web Application

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. It describes how BaseX can be used to both provide simple APIs and build complex web applications.

Startup

With Version 10, the default admin password has been removed, and the default ports have been changed from 8984/8985 to 8080/8081.

- Run one of the basexhttp or basexhttp.bat scripts. Call the script with the stop keyword to gracefully shut down the server.
- If you have installed BaseX on Windows, click on the BaseX HTTP Server (Start) icon.

Both an instance of Jetty and BaseX Database Server will be started. By default, Jetty listens to the port 8080, and the BaseX Database Server is accessible on port 1984. After startup, you can access a plain HTML welcome page via http://localhost:8080.

Unless you have already chosen an admin password yet (e.g., via the Windows installer or a previous installation), you can do so by invoking the PASSWORD command on your terminal:

```
basexhttp -c PASSWORD
```

```
BaseX [HTTP Server]

[main] INFO org.eclipse.jetty.util.log - Logging initialized @239ms to org.eclipse.jetty.util.log.Slf4jLog

HTTP STOP Server was started (port: 8081).
HTTP Server was started (port: 8080).
Password: _
```

The Jetty logging level can be adjusted by adding the following properties to the start script:

```
-Dorg.eclipse.jetty.util.log.class=org.eclipse.jetty.util.log.Slf4jLog
-D{classref}.LEVEL=DEBUG
```

Various command-line options are available to simplify batch processing. The start script can be adjusted for individual purposes (e.g. if the default memory limit is too restrictive).

BaseX can also be deployed as web servlet in a servlet container or with Maven:

Servlet Container

In order to deploy BaseX HTTP Services in a servlet container, you can download the WAR distribution of BaseX from the download site, or compile it by calling mvn compile war:war in the basex-api directory. The WAR file can then be deployed following the instructions of the corresponding servlet container (Jetty, Tomcat, etc.).

You can configure the port, context path, etc. by following the instructions of the corresponding 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/basex/rest).

If you use Jetty (which is the default HTTP server of BaseX), the server configuration is available via the jetty.xml file, which is stored in the WEB-INF directory next to the web.xml. For detailed configuration, refer to the Jetty Documentation.

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

Check out the BaseX sources via Eclipse or Git. Execute `mvn install` in the main project directory and then `mvn install jetty:run` in the `basex-api` subdirectory. This will start a Jetty instance in which the servlets will be deployed.

The same options as in the case of deployment apply in a servlet container. 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 session in the `pom.xml` file.

Services

The following services are available and enabled by default:

<table>
<thead>
<tr>
<th>Name</th>
<th>Standard Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESTXQ</td>
<td>/</td>
<td>Write enriched APIs and full web applications with XQuery.</td>
</tr>
<tr>
<td>WebSockets</td>
<td>ws/</td>
<td>Bidirectional client/server communication.</td>
</tr>
<tr>
<td>REST</td>
<td>rest/</td>
<td>Straightforward access to XML databases and its resources.</td>
</tr>
<tr>
<td>WebDAV</td>
<td>webdav/</td>
<td>Database access via the file system.</td>
</tr>
<tr>
<td>Default</td>
<td>static/</td>
<td>Access to static server resources (HTML, JavaScript, CSS, images, ...).</td>
</tr>
</tbody>
</table>

The DBA is a web-based database administration interface written in RESTXQ. It allows you to create and administrate databases, evaluate queries in realtime, view log files, manage users, etc. It is embedded in the full distributions of BaseX, and it can be accessed after startup via `http://localhost:8080/dba/`.

Configuration

Unless BaseX is deployed as servlet, the location of the web application directory can be adjusted via the `WEBPATH` option, and compression of HTTP responses can be enabled via the `GZIP` option.

Further database options can be defined as context parameters in the `web.xml` file. The most important options for the web application context are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USER</td>
<td>admin</td>
<td>If a user is specified, no credentials must be passed on by the client.</td>
</tr>
<tr>
<td>HTTPLOCAL</td>
<td>false</td>
<td>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 <code>true</code>.</td>
</tr>
<tr>
<td>RESTXQPATH</td>
<td>.</td>
<td>Relative or absolute directory referencing the RESTXQ modules. By default, the option points to the standard web application directory.</td>
</tr>
<tr>
<td>RESTPATH</td>
<td>.</td>
<td>Relative or absolute directory referencing queries and command-scripts that can be invoked via the <code>run operation</code> of REST. By default, the option points to the standard web application directory.</td>
</tr>
<tr>
<td>AUTHMETHOD</td>
<td>Basic</td>
<td>The default authentication method proposed by the server. The available methods are <code>Basic</code> and <code>Digest</code>.</td>
</tr>
</tbody>
</table>

All options are prefixed with `org.basex`. Local file paths in options may be absolute or relative. If a relative path is specified, its root will be the servlet's (webapp) path:
Context parameters can be requested from XQuery via `proc:property-names` and `proc:property`. How to set these options is specific to the servlet container. For example, in Jetty it can be 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.

To enable or disable a specific service, the corresponding servlet entry in the `web.xml` file needs to be removed/ commented.

**Authentication**

No credentials need to be supplied if a default user is assigned to a service in the `web.xml` file. In the following example, the user `rest-user` is specified for the REST service:

```xml
<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>
</servlet>
```

If the HTTP server is started with no pre-defined user, the credentials must be passed on by the client via Basic Authentication or Digest Authentication, depending on the chosen authentication method in the configuration.

With cURL, internet browsers, and other tools, you can specify basic authentication credentials within the request string as plain text, using the format `USER:PASSWORD@URL`:

```
http://admin:...@localhost:8080/
```

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

**Changelog**

Version 10.0

- Updated: The default admin password has been removed, and the default ports have been changed from 8984/8985 to 8080/8081.

Version 9.0

- Updated: `jetty.xml` configuration file (required for Jetty 9).

Version 8.6

- Updated: Authentication readded to RESTXQ.
- Updated: No password must be specified in the `web.xml` file anymore.
- Updated: Server-side user and authentication method is now enforced (cannot be overwritten by client).
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 10. 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 browser-based database administration interface, the DBA. It allows you to create and administer databases, evaluate queries in real time, view log files, monitor logged-in users, manage users, etc. The server-side code is completely written in XQuery and RESTXQ.

These were our design goals:

• The code base is supposed to inspire and motivate you developing your own RESTXQ web applications.

• The XQuery DBA code is very lean; it consumes less than 150 KB. It uses plain and simple JavaScript and no framework.

• 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 subdirectory 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.

If you put DBA online along with your web page, please ensure at the very least…

• that you have chosen a strong password for your admin user, and

• that the BaseX process has not been started with admin privileges.

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

On the welcome page, you need to authenticate yourself by entering the name and password of a user with admin permissions.

Logs

All database logs are listed, ordered by creation time in descending order. The interactive filter allows you to search in the users and text columns via regular expressions. The found substrings are highlighted in the output.

For each day, a new log file is created. Old log files can be selected and deleted.

Databases

The database panel contains a list of all databases. Databases can be created, optimized and dropped. If a database is selected, the database resources, backups and properties are listed. Queries can be run on single database resources.
Queries

XQuery expressions can be run in the Queries panel. If evaluation takes too long, or if it consumes too much memory, the execution will be interrupted. You can choose if your query is updating or not.

Inside the editor area, you can press Ctrl-Enter to execute the query. You can press Shift-Ctrl-Enter to run your XQuery expression as updating query (or non-updating, if “Updating” is chosen in the dropdown menu).

Existing queries can be opened, and saved for future operations. All files will be stored in the current DBA working directory.

Files

Remote files can be downloaded, opened in the query editor or run as BaseX jobs, and new files can be uploaded. The chosen directory affects the Queries panel.

You can edit your RESTXQ code in real time by switching to the RESTXQ or repository directory and opening the corresponding modules.

Jobs

In the Jobs panel, all queries are listed that are currently being run or queued. You can view details on particular jobs, spot potential bottlenecks, or spot and terminate malicious requests. The panel will always list at least one job, which is the one that is currently preparing your HTTP response.

Users

Existing users can be updated and new users can be created. Extra information can be viewed and modified both globally and locally (see User Management for more information).
Sessions

The Web Sessions table lists all users that are currently registered in an application or the DBA. See the DBA RESTXQ code for information on how clients can be registered and logged out.

The Database Sessions table shows clients that are connected via the client/server architecture.

Settings

In the settings, you can tweak some DBA options, enforce a garbage collection and view all current global and local databases options.

Changelog

Version 9.4

• Updated: Logging was improved for millions of log entries

Version 8.6

• Updated: Always accessible, even if job queue is full

• Removed: Remote connections (to allow for better optimizations and less locking)

Version 8.4

• Added: Editor: Key combination 'Shift-Ctrl-Enter', real time mode removed.

Introduced with Version 8.0.
Part III. Command Line
Chapter 11. Command-Line Options

Read this entry online in the BaseX Wiki.

This article is part of the Getting Started Guide. Each BaseX Startup mode has one or more command-line options which are described in this article.

Command-line options can be specified multiple times. Please note that all options will be evaluated in the given order. The standard input can be parsed by specifying a single dash (−) as argument.

Standalone

The following options are available for the standalone Command-Line Client:

$ basex -h
BaseX [Standalone]
Usage: basex [-bcdiLoqrRstuvVwxz] [input]
[input]    XQuery or command file, or query string
-b<args>   Bind external query variables
-c<input>  Execute commands from file or string
-d         Toggle debugging output
-i<input>  Bind file or database to context
-I<input>  Bind input string to context
-o<path>   Write output to local file
-q<expr>   Execute XQuery expression
-r<num>    Run query multiple times
-R         Toggle query execution
-s<args>   Set serialization parameters
-t[path]   Run tests in file or directory
-u         Toggle updates in original files
-v         Toggle output of progress info
-V         Toggle detailed query output
-w         Toggle whitespace stripping
-x         Toggle output of query plan
-z         Toggle output of query result

Further details are 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, it might be necessary to escape some characters depending on your operating system.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Option</th>
<th>Default</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>[input]</td>
<td>Evaluates the specified input:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The input string may point to an existing file. If the file suffix is .bxs, the file contents will be evaluated as Command Script; any other file content will be evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Otherwise, the input string itself is evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-b&lt;args&gt;</td>
<td>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.</td>
<td>BINDINGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• bv=example &quot;declare variable $v external; $v&quot;.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• b(URL)ln=value&quot;declare namespace ns='URL';</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Command-Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c &lt;input&gt;</code></td>
<td>Executes commands. If the specified input is a valid URI or file reference, this file will be evaluated as Command Script.</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Toggles the debugging mode. Debugging information is output to standard error.</td>
</tr>
<tr>
<td><code>-i &lt;input&gt;</code></td>
<td>Opens the specified XML file, directory with XML files, or database. The opened input can then be processed by a command or XQuery expression.</td>
</tr>
<tr>
<td><code>-I &lt;input&gt;</code></td>
<td>Assigns an input string as item of type <code>xs:untypedAtomic</code> to the query context.</td>
</tr>
<tr>
<td><code>-o &lt;path&gt;</code></td>
<td>All command and query output is written to the specified file.</td>
</tr>
<tr>
<td><code>-q &lt;expr&gt;</code></td>
<td>Executes the specified string as XQuery expression.</td>
</tr>
<tr>
<td><code>-r &lt;num&gt;</code></td>
<td>Specifies how often a specified query will be evaluated.</td>
</tr>
<tr>
<td><code>-R</code></td>
<td>Specifies if a query will be evaluated or parsed and compiled only.</td>
</tr>
<tr>
<td><code>-s &lt;args&gt;</code></td>
<td>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 (=).</td>
</tr>
<tr>
<td><code>-t [path]</code></td>
<td>Runs all Unit tests in the specified file or directory.</td>
</tr>
<tr>
<td><code>-u</code></td>
<td>Propagates updates on input files back to disk.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Toggles the output of process and timing information.</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Prints detailed query information to the standard output, including details on the compilation and profiling steps.</td>
</tr>
<tr>
<td><code>-w</code></td>
<td>Toggles whitespace stripping of XML text nodes. By default, whitespaces will be preserved.</td>
</tr>
<tr>
<td><code>-x</code></td>
<td>Toggles the output of the query execution plan, formatted as XML.</td>
</tr>
<tr>
<td><code>-z</code></td>
<td>Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ns:ln</code></td>
<td>Declares variable $ns:ln external; $ns:ln&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-c list.</code></td>
<td><code>-c commands.txt</code></td>
</tr>
</tbody>
</table>
| `*` | `-c"<info/">

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DEBUG</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>iitems.xml</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>false</code></td>
<td></td>
</tr>
</tbody>
</table>
GUI

The following options are available for the standalone Graphical User Interface:

```bash
$ basexgui -h
BaseX [GUI]
Usage: basexgui [-d] [files]
   [files] Open specified files
   -d Enable debugging
```

You can pass one or more files as parameters. If an XML document is specified, a database instance can be created from this file. Other files are opened in the editor.

Server

The following options are available for the Database Server:

```bash
$ basexserver -h
BaseX [Server]
Usage: basexserver [-cnpSz] [stop]
   stop       Stop running server
   -c<input>  Execute commands from file or string
   -d         Enable debugging output
   -n<name>   Set host the server is bound to
   -p<port>   Set server port
   -S         Start as service
   -z         Suppress logging
```

Details on all options are listed in the following table (equivalent database options are shown in the table as well).

For the examples to work, it might be necessary to escape some characters depending on your operating system.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Option</th>
<th>Default</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>Stops a local database server instance and quits.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-c&lt;input&gt;</td>
<td>Executes commands from file or string. If the specified input is a valid URI or file reference, this file will be evaluated as Command Script.</td>
<td></td>
<td></td>
<td>-c&quot;open database;info&quot;</td>
</tr>
<tr>
<td>-d</td>
<td>Enables debugging output. Debugging information is output to standard error.</td>
<td>DEBUG</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>-n&lt;name&gt;</td>
<td>Specifies the host the server will be bound to.</td>
<td>SERVERHOST</td>
<td></td>
<td>-p127.0.0.1</td>
</tr>
<tr>
<td>-p&lt;port&gt;</td>
<td>Specifies the port on which the server will be addressable.</td>
<td>SERVERPORT</td>
<td>1984</td>
<td>-p9999</td>
</tr>
<tr>
<td>-S</td>
<td>Starts the server as service (i.e., in the background). Use YAJSW, or start BaseX as an ordinary background process to get more options.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-z</td>
<td>Prevents the generation of log files.</td>
<td>LOG</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

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

If the Database Client is launched, you will be requested for a username and password.

```bash
$ basexclient -h
BaseX [Client]
```
Usage: basexclient [-bcdiInopRrsUvVwxz] [input]

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Option</th>
<th>Default</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>[input]</td>
<td>Evaluates the specified input:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The input string may point to an existing file. If the file suffix is .bxs, the file contents will be evaluated as Command Script; any other file content will be evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Otherwise, the input string itself is evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-b&lt;args&gt;</td>
<td>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 or Expanded QName Notation.</td>
<td>BINDINGS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-c&lt;input&gt;</td>
<td>Executes commands. If the specified input is a valid URI or file reference, its content will be executed instead. Empty lines and lines starting with the number sign # will be ignored.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-d</td>
<td>Toggles the debugging mode. Debugging information is output to standard error.</td>
<td>DEBUG</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>-i&lt;input&gt;</td>
<td>Opens the specified XML file, directory with XML files, or database. The opened input</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Command-Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-i</code>&lt;input&gt;</td>
<td>Assigns an input string as item of type <code>xs:untypedAtomic</code> to the query context.</td>
<td><code>-i &quot;Hello Universe&quot; -q &quot;.&quot;</code></td>
</tr>
<tr>
<td><code>-n</code>&lt;name&gt;</td>
<td>Specifies the host name on which the server is running.</td>
<td>HOST <code>localhost</code> <code>-nserver.basex.org</code></td>
</tr>
<tr>
<td><code>-o</code>&lt;path&gt;</td>
<td>All command and query output is written to the specified file.</td>
<td></td>
</tr>
<tr>
<td><code>-p</code>&lt;port&gt;</td>
<td>Specifies the port on which the server is running.</td>
<td>PORT <code>1984</code> <code>-p9999</code></td>
</tr>
<tr>
<td><code>-P</code>&lt;pass&gt;</td>
<td>Specifies the user password. If this flag is omitted, the password will be requested on command line. <em>Warning:</em> When the password is supplied with this flag, it may end up in logs or the bash history.</td>
<td>PASSWORD <code>admin</code> <code>-P...</code></td>
</tr>
<tr>
<td><code>-q</code>&lt;expr&gt;</td>
<td>Executes the specified string as XQuery expression.</td>
<td><code>-q&quot;1+2&quot;</code></td>
</tr>
<tr>
<td><code>-r</code>&lt;num&gt;</td>
<td>Specifies how often a specified query will be evaluated.</td>
<td>RUNS <code>1</code> <code>-V -r10 &quot;1&quot;</code></td>
</tr>
<tr>
<td><code>-R</code></td>
<td>Specifies if a query will be executed or parsed only.</td>
<td>RUNQUERY <code>true</code> <code>-V -R &quot;1&quot;</code></td>
</tr>
<tr>
<td><code>-s</code>&lt;args&gt;</td>
<td>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 (<code>=</code>).</td>
<td>SERIALIZER <code>smethod=text</code></td>
</tr>
<tr>
<td><code>-U</code>&lt;name&gt;</td>
<td>Specifies the username. If this flag is omitted, the username will be requested on command line.</td>
<td>USER <code>admin</code> <code>-Uadmin</code></td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Prints process and timing information to the standard output.</td>
<td>false</td>
</tr>
<tr>
<td><code>-V</code></td>
<td>Prints detailed query information to the standard output, including details on the compilation and profiling steps.</td>
<td>QUERYINFO <code>false</code></td>
</tr>
<tr>
<td><code>-w</code></td>
<td>Toggles whitespace stripping of XML text nodes. By default, whitespaces will be preserved.</td>
<td>STRIPWS <code>false</code></td>
</tr>
<tr>
<td><code>-x</code></td>
<td>Toggles the output of the query execution plan, formatted as XML.</td>
<td>XMLPLAN <code>false</code></td>
</tr>
<tr>
<td><code>-z</code></td>
<td>Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.</td>
<td>SERIALIZE <code>true</code></td>
</tr>
</tbody>
</table>

## HTTP Server

*With Version 10, the default ports have been changed from 8984/8985 to 8080/8081.*

The following options are available for the **HTTP Server**:

```
$ basexhttp -h
BaseX [HTTP]
Usage: basexhttp [-cdlnpsSUz] [stop]
```
Command-Line Options

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Option</th>
<th>Default</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>stop</td>
<td>Stop running server</td>
<td>pom.xml</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-c&lt;input&gt;</td>
<td>Execute commands from file or string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-d</td>
<td>Enable debugging output</td>
<td>DEBUG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-g</td>
<td>Enable GZIP support</td>
<td>GZIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-h&lt;port&gt;</td>
<td>Set port of HTTP server</td>
<td>jetty.xml</td>
<td>8080</td>
<td>-h9999</td>
</tr>
<tr>
<td>-l</td>
<td>Start in local mode</td>
<td>HTTPLOCAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-n&lt;name&gt;</td>
<td>Set host name of database server</td>
<td>HOST</td>
<td>localhost</td>
<td>-nservice.basex.org</td>
</tr>
<tr>
<td>-p&lt;port&gt;</td>
<td>Set port of database server</td>
<td>SERVERPORT</td>
<td>1984</td>
<td>-p9998</td>
</tr>
<tr>
<td>-s&lt;port&gt;</td>
<td>Specify port to stop HTTP server</td>
<td>STOPPORT</td>
<td>orpom.xml</td>
<td>8081</td>
</tr>
<tr>
<td>-S</td>
<td>Start as service</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-U&lt;name&gt;</td>
<td>Specify username</td>
<td>USER</td>
<td></td>
<td>-Uadmin</td>
</tr>
<tr>
<td>-z</td>
<td>Suppress logging</td>
<td>LOG</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The meaning of all options is listed in the following table (equivalent database options are shown in the table as well). For the examples to work, it might be necessary to escape some characters depending on your Operating System.

Changelog

Version 10.0
• Updated: Whitespaces are now preserved by default (see STRIPWS for more details).

Version 9.0
• Added: BaseXHTTP, command-line option -c.
Command-Line Options

• Updated: BaseXHTTP, command-line option -c, additionally accepts valid URLs and file references.

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 12. Start Scripts

Read this entry online in the BaseX Wiki.

BaseX can be started in different ways. The Windows and ZIP distributions include various start scripts, which are presented in the following, and which can also be maintained separately.

- We recommend you to manually add the `bin` directory of your BaseX instance to the `PATH` variable of your environment.
- If you use the Windows installer, that’s done automatically.
- You can copy the start scripts to another location in your file system. After that, you should edit the scripts and assign the BaseX directory to the `MAIN` variable.
- If you are compiling the source code with Maven, you can launch BaseX via the scripts in the `basex-core/etc` and `basex-api/etc` subdirectories of the project.

If BaseX terminates with an `Out of Memory` or `Java heap space` error, you can assign more RAM via the `-Xmx` flag (see below). A conservative value was chosen in our distributions to ensure that BaseX will also run on older JVMs.

**Standalone**

Use the following scripts to launch the standalone version of BaseX:

**Windows: basex.bat**

```bash
@echo off
setLocal EnableDelayedExpansion
REM Path to core and library classes
set MAIN=%~dp0/..
set CP=%MAIN%/BaseX.jar;%MAIN%/lib/*;%MAIN%/lib/custom/*
REM Options for virtual machine
set BASEX_JVM=-Xmx1200m %BASEX_JVM%
REM Run code
java -cp "%CP%" %BASEX_JVM% org.basex.BaseX %*
```

**Linux/Mac: basex**

```bash
#!/usr/bin/env bash
# Path to this script
FILE="$(BASH_SOURCE[0])"
while [ -h "$FILE" ] ; do
do $(cd -P "$FILE" || cd -P "$(dirname "$FILE")/.." && cd -P "$(dirname "$SRC")" && pwd )/$(basename "$SRC")
done
MAIN="$( cd -P "$dirmame "$FILE"/.." && pwd )"
# Core and library classes
CP=$MAIN/BaseX.jar:$MAIN/lib/*:$MAIN/lib/custom/*:$CLASSPATH
# Options for virtual machine (can be extended by global options)
BASEX_JVM="-Xmx2g $BASEX_JVM"
# Run code
```

45
Java -cp "$CP" $BASEX_JVM org.basex.BaseX "$@"

**GUI, Server, Client**

If you would like to launch the GUI, Server or Client version of BaseX, please replace the class name in org.basex.BaseX with either BaseXGUI, BaseXServer or BaseXClient.

**HTTP Server**

The scripts for running Web Applications can be found below:

**Windows:** basexhttp.bat

```bash
@echo off
setLocal EnableDelayedExpansion

REM Path to core and library classes
set MAIN=%~dp0/..
set CP=%MAIN%/BaseX.jar;%MAIN%/lib/*;%MAIN%/lib/custom/*

REM Options for virtual machine
set BASEX_JVM=-Xmx1200m %BASEX_JVM%

REM Run code
java -cp "$CP" %BASEX_JVM% org.basex.BaseXHTTP %*
```

**Linux/Mac:** basexhttp

```bash
#!/usr/bin/env bash

# API, core, and library classes
CP=$MAIN/BaseX.jar:$MAIN/lib:*:$MAIN/lib/custom::*:$CLASSPATH

# Options for virtual machine (can be extended by global options)
BASEX_JVM=-Xmx2g $BASEX_JVM"

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

### Included Start Scripts

The BaseX Windows and ZIP distributions include the following start scripts:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux/Mac</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basex.bat</td>
<td>basex</td>
<td>Launches the BaseX standalone mode.</td>
</tr>
<tr>
<td>basexclient.bat</td>
<td>basexclient</td>
<td>Starts a BaseX client.</td>
</tr>
<tr>
<td>basexgui.bat</td>
<td>basexgui</td>
<td>Starts the BaseX GUI.</td>
</tr>
<tr>
<td>basexhttp.bat</td>
<td>basexhttp</td>
<td>Starts the BaseX HTTP Server.</td>
</tr>
<tr>
<td>basexserver.bat</td>
<td>basexserver</td>
<td>Starts the BaseX database server.</td>
</tr>
</tbody>
</table>
For the BaseX HTTP and database server, additional stop scripts are available:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux/Mac</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basexhttpstop.bat</td>
<td>basexhttpstop</td>
<td>Stops the BaseX HTTP Server.</td>
</tr>
<tr>
<td>basexserverstop.bat</td>
<td>basexserverstop</td>
<td>Stops the BaseX database server.</td>
</tr>
</tbody>
</table>

**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 IV. General Info
Chapter 13. 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. It may contain one or more resources, which are addressed by a unique database path. There is no explicit layer for collections: Instead, collections are implicitly created and deleted, and collections result from the existence of documents in specific paths.

As a single database is restricted to 2 billion XML nodes (see Statistics), but resources can easily be distributed across multiple database instances. Multiple databases can be addressed (queried, updated) by a single XQuery expression.

Three different resource types exist:

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Documents</td>
<td>The default resource type. The storage and index features are optimized for XML contents, or any other contents stored in an XML representation.</td>
</tr>
<tr>
<td>Binary Data</td>
<td>Binary data: Raw data of any type, stored in its binary representation. See Binary Data for more information.</td>
</tr>
<tr>
<td>XQuery Values</td>
<td>Introduced with Version 10: Results of XQuery expressions, stored in a binary representation for fast retrieval. All value types are supported, including maps and arrays, but excluding any other function items.</td>
</tr>
</tbody>
</table>

Create Databases

Databases can be created via Commands, via XQuery, in the GUI, and with various APIs. If an initial input is specified with a create operation, some time can be saved, as the specified resources will be added to the database in a bulk operation:

- **Command-Line**: `CREATE DB documents /path/to/resources`: Add resources in the specified path to a database named documents.

- **GUI**: Go to Database → New, press Browse… to choose an initial file or directory, and press OK.

The database name is composed of a restricted set of characters (see Valid Names). Various Parser can be selected to control the import process, or to convert data of different input type to XML.

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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>db:get</code></td>
<td><code>db:get(&quot;db&quot;, &quot;path/to/docs&quot;)</code></td>
<td>Returns all documents that are found in the database db at the (optional) path path/to/docs.</td>
</tr>
<tr>
<td><code>fn:collection</code></td>
<td><code>collection(&quot;db/path/to/docs&quot;)</code></td>
<td>Returns all documents at the location path/to/docs in the database</td>
</tr>
</tbody>
</table>
Databases

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn:doc</td>
<td>doc(&quot;db/path/to/doc.xml&quot;)</td>
<td>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.</td>
</tr>
</tbody>
</table>

You can access multiple databases in a single query:

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

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

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

- **db:name**

  ```xml```
  db:name($node)
  ```xml```

  Returns the name of the database in which the specified $node is stored.

- **db:path**

  ```xml```
  db:path($node)
  ```xml```

  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:

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

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.

**Binary Data**

The BINARY GET command and the db:get-binary function can be used to return files in their native byte representation.

If the API you use does not support binary output (which is e.g. the case for various Client language bindings), you can convert your binary data to its string representation before returning it to the client:

```xml```
string(db:get-binary('multimedia', 'sample.avi'))
```xml```
**XQuery Values**

With `db:get-value`, XQuery values can be retrieved. In the following example, we assume that an XQuery map `cities` was stored in an `indexes` database:

```xml
let $city-map := db:get-value('indexes', 'cities')
return $city-map?Chile
```

**Update Resources**

**Commands**

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

- XML documents can be added with the **PUT** and **ADD** commands.
- Binary data is stored with **BINARY PUT**.
- Resources of all types can be deleted via **DELETE**.

**AUTOFLUSH** can be turned off before *bulk operations* (i.e., before numerous new resources are added to the database).

If **ADDCACHE** is enabled, the input will be cached before it is added to the database. This is helpful when the input documents are expected to consume too much main-memory.

With the following **command script**, an empty database is created, two resources are added (one directly, another one cached), and all data is exported to the file system:

```xml
CREATE DB example
SET AUTOFLUSH false
ADD example.xml
SET ADDCACHE true
ADD /path/to/xml/documents
BINARY PUT TO images/ 123.jpg
EXPORT /path/to/file-system/
```

**XQuery**

You can also use functions from the Database Module to add, replace, or delete XML documents:

```xml
db:add('documents', '/path/to/xml/resources/')
```

Function from other modules, such as the File Module, can be utilized to filter the input. With the following code, all files that contain numbers in the filename are selected, and stored as XML. If an input file contains no well-formed XML, it is stored as binary resource, and the error message is stored as a string value:

```xml
let $db := 'documents'
let $root := '/path/to/resources/'
for $path in file:list($root)
where matches($path, '\d+')
return try {
    db:put($db, fetch:doc($root || $path), $path)
} catch * {
    db:put-binary($db, $root || $path, $path),
    db:put-value($db, $err:description, $path || '.error')
}
```

The error messages can e.g. be analyzed in a second step:

```xml
let $errors := db:get-value('documents')
for $filename in map:keys($errors)
where ends-with($filename, '.error')
```
Export Database

All resources stored in a database can be exported, i.e., written back to disk, e.g., as follows:

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

Main-Memory Databases

A database can be created in main-memory by enabling the `MAINMEM` option. Next, in the standalone context, a main-memory database can be created, which can then be accessed by subsequent commands.

If a BaseX server is started, and if a database is created in its context at startup time, e.g., with the command-line option `-c` and a CREATE DB call, BaseX clients can then access and update this database:

```bash
# Server
basexserver -c"SET mainmem on" -c"CREATE DB mainmem document.xml"
BaseX [Server]
Server was started (port: 1984).
MAINMEM: true
Database 'mainmem' created in 1782.80 ms.

# Client
basexclient
Username: ...
Password: ...
BaseX [Client]
Try 'help' to get more information.
> XQUERY count(db:get('mainmem')//*)
1876462
Query executed in 0.97 ms.
```

Additional notes:

- You can force an ordinary database, or parts of it, to being temporarily copied to memory by applying an empty main-memory update on a database node: `db:get('some-db') update { }`
- If you open local or remote documents with `fn:doc` or `fn:collection`, the resulting internal representation is identical to those of main-memory database instances (regardless of which value is set for `MAINMEM`).

Changelog

Version 10.0
- Added: New resource type for XQuery values.

Version 8.4
- Updated: Items of binary type can be output without specifying the obsolete raw serialization method.

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 14. 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 be executed from the Command Line, as part of Scripts, via 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

On command line, multiple commands can be written down in a single line (separated by semicolons). You can also put them into a command script: Database commands in both string and XML syntax can be placed in a text file and stored as file with the BaseX command script suffix .bxs. If the path to a script file is passed on to BaseX on command-line, or if it is opened in the GUI editor, it will be recognized and evaluated as such.

String Syntax

Lines starting with # are interpreted as comments and are skipped. With the following script, a database is created, two documents are added to it, and a query is performed:

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

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.

The XML syntax provides more flexibility here. Multiple commands can be enclosed by a <commands/> root element. Some commands, such as ADD, allow you to directly embed XML documents. If you want to embed XML in XQuery expressions, entities should be encoded, or the CDATA syntax should be used:

```xml
<commands>
  <create-db name='test'/>
  <add path='embedded.xml'><root>embedded</root></add>
  <!-- run query -->
  <xquery><![CDATA[
    <hits>{ count(//text()) } </hits>
  ]]> </xquery>
  <close/>
</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.
Commands

• X*, Y*, Z* addresses all names starting with the characters X, Y, or Z.

Valid Names

Names of databases and users follow the same constraints: Names must at least have one printable character, including letters, numbers, and any of the special characters !#%&'()+-=[]^_`{}~. The following characters are disallowed:

• *, ?*: glob syntax
• ;: Separator for multiple database commands on the command line
• \/: Directory path separators
• :*<>|: invalid filename characters on Windows
• Names starting or ending with .: hidden folders (e.g. the .logs directory)

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 can be a file or directory path to XML documents, a remote URL, or a string containing XML:

• name must be a valid database name
• database creation can be controlled by setting Create Options

If you need to add initial resources, it is always faster to supply them at creation time than adding them in a subsequent step via ADD.

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 and 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

Updated with Version 10:path argument dropped.
## Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Syntax</th>
<th>XML Syntax</th>
<th>Permission</th>
<th>Summary</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPEN</strong></td>
<td>OPEN [name]</td>
<td>&lt;open name='...'/&gt;</td>
<td><em>READ</em></td>
<td>Opens the database specified by <em>name</em>.</td>
<td>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.</td>
</tr>
<tr>
<td><strong>CHECK</strong></td>
<td>CHECK [input]</td>
<td>&lt;check input='...'/&gt;,</td>
<td><em>READ/CREATE</em></td>
<td>This convenience command combines OPEN and CREATE DB: If a database with the name <em>input</em> exists, and if there is no existing file or directory with the same name that has a newer timestamp, the database is opened. Otherwise, a new database is created; if the specified input points to an existing resource, it is stored as initial content.</td>
<td>The command fails if the addressed database could neither be opened nor created.</td>
</tr>
<tr>
<td><strong>CLOSE</strong></td>
<td>CLOSE</td>
<td>&lt;close/&gt;,</td>
<td><em>READ</em></td>
<td>Closes the currently opened database.</td>
<td>The command fails if the database files could not be closed for some reason.</td>
</tr>
<tr>
<td><strong>LIST</strong></td>
<td>LIST ([name] ([path]))</td>
<td>&lt;list (name='...' (path='...'))/&gt;,</td>
<td><em>NONE</em></td>
<td>Lists all available databases. If <em>name</em> is specified, the resources of a database are listed. The output can be further restricted to the resources matching the specified <em>path</em>. If database resources are listed, the size is either the number of nodes (for XML resources) or the number of bytes (for binary resources).</td>
<td>The command fails if the optional database cannot be opened, or if the existing databases cannot be listed for some other reason.</td>
</tr>
<tr>
<td><strong>DIR</strong></td>
<td>DIR ([path])</td>
<td>&lt;dir (path='...')/&gt;,</td>
<td><em>READ</em></td>
<td>Lists directories and resources of the currently opened database and the specified <em>path</em>. If database resources are listed, the size is either the number of nodes (for XML resources) or the number of bytes (for binary resources).</td>
<td></td>
</tr>
</tbody>
</table>

*Introduced with Version 10.*
## EXPORT

<table>
<thead>
<tr>
<th>Syntax</th>
<th>EXPORT [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;export path='...'/&gt;</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
</tbody>
</table>

**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|TOKEN|FULLTEXT] |
|------------|-------------|
| XML Syntax | `<create-index type='text|attribute|token|fulltext'/>`, |
| Permission | WRITE |

**Summary**
Creates the specified Value Index. The current Index Options will be considered when creating the 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|TOKEN|FULLTEXT] |
|------------|-------------|
| XML Syntax | `<drop-index type='text|attribute|token|fulltext'/>`, |
| Permission | WRITE |

**Summary**
Drops the specified Value 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.

## ALTER DB

<table>
<thead>
<tr>
<th>Syntax</th>
<th>ALTER DB [name] [newname]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;alter-db name='...' newname='...'/&gt;</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
</tbody>
</table>

**Summary**
Renames the database specified by `name` to `newname`. `newname` must be a valid database name.

**Errors**
The command fails 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>DROP DB [name]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;drop-db name='...'/&gt;</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
</tbody>
</table>

**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.
## Commands

### COPY

<table>
<thead>
<tr>
<th>Syntax</th>
<th>COPY [name] [newname]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;copy name='...' newname='...'/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
<tr>
<td>Summary</td>
<td>Creates a copy of the database specified by name. newname must be a valid database name.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the source database does not exist.</td>
</tr>
</tbody>
</table>

### INSPECT

<table>
<thead>
<tr>
<th>Syntax</th>
<th>INSPECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;inspect/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>READ</td>
</tr>
<tr>
<td>Summary</td>
<td>Performs some integrity checks on the opened database and returns a brief summary.</td>
</tr>
</tbody>
</table>

### Backups

*Introduced with Version 10:* Support for general data (registered users, scheduled services and key-value stores).

### CREATE BACKUP

*Updated with Version 10:* Support for comments.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>CREATE BACKUP ([name] ([comment]))</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;create-backup (name='...') (comment='...')/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
<tr>
<td>Summary</td>
<td>Creates a backup of the database specified by name, with an optional comment. If no name is supplied, general data will be backed up. 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.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified database does not exist, or if it could not not be zipped for some other reason.</td>
</tr>
<tr>
<td>Examples</td>
<td>• BACKUP db, creates a zip archive of the database db (e.g. db-2014-04-01-12-27-28.zip) in the database directory.</td>
</tr>
</tbody>
</table>

### DROP BACKUP

<table>
<thead>
<tr>
<th>Syntax</th>
<th>DROP BACKUP ([name])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;drop-backup (name='...')/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
<tr>
<td>Summary</td>
<td>Drops all backups of the database with the specified name, or a specific backup file if the name ends with its timestamp. If no name is supplied, backups with general data are addressed. The Glob Syntax can be used to address more than one database.</td>
</tr>
<tr>
<td>Examples</td>
<td>• DROP BACKUP abc*, deletes the backups of all databases starting with the characters abc.</td>
</tr>
<tr>
<td></td>
<td>• DROP BACKUP factbook-2021-05-16-13-13-10, deletes a specific backup file.</td>
</tr>
</tbody>
</table>

### ALTER BACKUP

<table>
<thead>
<tr>
<th>Syntax</th>
<th>ALTER BACKUP [name] [newname]</th>
</tr>
</thead>
</table>
Commands

**XML Syntax** `<alter-backup name='...' newname='...'/>`,

**Permission** `CREATE`

**Summary** Renames all backups of the database with the specified name to newname. If the name ends with a timestamp, only the specified backup file will be renamed. The **Glob Syntax** can be used to address more than one database.

**Examples**
- `ALTER BACKUP logs logs-backup`, renames the backups of the logs database to logs-backup.

### 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. If no name is supplied, general data will be restored. If general data is restored, it will only be available after BaseX has been restarted.

**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.

### SHOW BACKUPS

**Syntax** `SHOW BACKUPS`

**XML Syntax** `<show-backups/>,`

**Permission** `CREATE`

**Summary** Shows all database backups.

### Querying

**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 and XQUERY 1 to 10`, returns the results after having run the query 10 times.
- `SET XMLPLAN true and XQUERY 1 to 10`, returns the result and prints the query plan as XML.

### GET

*Introduced with Version 10.* The old GET command has been renamed to **SHOW** OPTIONS.

**Syntax** `GET [path]`

**XML Syntax** `<get path='...'/>,`

**Permission** `READ`
**Summary**  Retrieves an XML document from the opened database at the specified path.

**Errors**  The command fails if no database is opened or if the source path is invalid.

**BINARY GET**

*Updated with Version 10: Renamed (before: RETRIEVE).*

**Syntax**  BINARY GET [path]

**XML Syntax**  `<binary-get path='...'/>`.

**Permission**  READ

**Summary**  Retrieves a binary resource from the opened database at the specified path.

**Errors**  The command fails if no database is opened or if the source path is invalid.

**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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>REPO LIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;repo-list/&gt;.</td>
</tr>
<tr>
<td>Permission</td>
<td>READ</td>
</tr>
<tr>
<td>Summary</td>
<td>Lists all installed packages.</td>
</tr>
</tbody>
</table>

### REPO DELETE

<table>
<thead>
<tr>
<th>Syntax</th>
<th>REPO DELETE [name]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;repo-delete name='...'/&gt;.</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
<tr>
<td>Summary</td>
<td>Deletes the specified package with the specified name. What is called &quot;name&quot; can also be the id (which is the name followed by the version) or the directory of the package.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the package to be deleted is required by another package.</td>
</tr>
</tbody>
</table>

### Updates

### ADD

<table>
<thead>
<tr>
<th>Syntax</th>
<th>ADD (TO [path]) [input]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;add (path='...')&gt;[input]&lt;/add&gt;.</td>
</tr>
<tr>
<td>Permission</td>
<td>WRITE</td>
</tr>
</tbody>
</table>
| 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, the PUT command 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>DELETE [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;delete path='...'/&gt;.</td>
</tr>
</tbody>
</table>
Commands

<table>
<thead>
<tr>
<th>Permissions</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Deletes all documents from the currently opened database that start with the specified path.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened.</td>
</tr>
</tbody>
</table>

**RENAME**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>RENAME [path] [newpath]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;rename path='...' newpath='...'/&gt;</td>
</tr>
<tr>
<td>Permissions</td>
<td>WRITE</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened, or if the target path is empty.</td>
</tr>
<tr>
<td>Examples</td>
<td>• RENAME one.xml two.xml, renames the document one.xml to two.xml.</td>
</tr>
<tr>
<td></td>
<td>• RENAME / TOP, moves all documents to a TOP root directory.</td>
</tr>
</tbody>
</table>

**PUT**

*Updated with Version 10: Renamed (before: REPLACE).*

<table>
<thead>
<tr>
<th>Syntax</th>
<th>PUT [path] [input]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;put path='...'&gt;[input]&lt;/put&gt;</td>
</tr>
<tr>
<td>Permissions</td>
<td>WRITE</td>
</tr>
<tr>
<td>Summary</td>
<td>Adds or replaces resources in the currently opened database, addressed by path, with the file, directory or XML string specified by input.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened or if the specified path is invalid.</td>
</tr>
<tr>
<td>Examples</td>
<td>• PUT one.xml input.xml, replaces one.xml with the contents of the file input.xml.</td>
</tr>
<tr>
<td></td>
<td>• PUT top.xml &lt;xml/&gt;, replaces top.xml with the XML document &lt;xml/&gt;.</td>
</tr>
</tbody>
</table>

**BINARY PUT**

*Updated with Version 10: Renamed (before: STORE).*

<table>
<thead>
<tr>
<th>Syntax</th>
<th>BINARY PUT (TO [path]) [input]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;binary-put (path='...')&gt;[input]&lt;/store&gt;</td>
</tr>
<tr>
<td>Permissions</td>
<td>WRITE</td>
</tr>
<tr>
<td>Summary</td>
<td>Stores a binary resource specified via input in the opened database to the specified path:</td>
</tr>
<tr>
<td></td>
<td>• The input may either be a file reference, a remote URL, or a plain string.</td>
</tr>
<tr>
<td></td>
<td>• If the path denotes a directory, it needs to be suffixed with a slash (/).</td>
</tr>
<tr>
<td></td>
<td>• An existing resource will be replaced.</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

**OPTIMIZE**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>OPTIMIZE (ALL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;optimize/&gt;, &lt;optimize-all/&gt;</td>
</tr>
</tbody>
</table>
Commands

<table>
<thead>
<tr>
<th>Permission</th>
<th>WRITE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Optimizes the index structures, metadata and statistics of the currently opened database:</td>
</tr>
<tr>
<td></td>
<td>• If ALL is specified, all database structures are completely reconstructed. The database size will be reduced, and all orphaned data will be deleted.</td>
</tr>
<tr>
<td></td>
<td>• Without ALL, only the outdated index structures and database statistics will be updated. If the database is completely up-to-date, nothing will be done.</td>
</tr>
<tr>
<td></td>
<td>• Database options will be adopted from the original database. Only AUTOOPTIMIZE and (if ALL is specified) UPDINDEX will be adopted from the current options.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened, or if the currently opened database is a main-memory instance.</td>
</tr>
</tbody>
</table>

**FLUSH**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>FLUSH</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;flush/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>WRITE</td>
</tr>
<tr>
<td>Summary</td>
<td>Explicitly flushes the buffers of the currently opened database to disk. This command is applied if AUTOFLUSH has been set to false.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened.</td>
</tr>
</tbody>
</table>

**User Management**

**CREATE USER**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>CREATE USER [name] ([password])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;create-user name='...'&gt;([password])&lt;/create-user&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>ADMIN</td>
</tr>
<tr>
<td>Summary</td>
<td>Creates a user with the specified name and password. If no password is specified, it is requested via the chosen frontend (GUI or bash).</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified user already exists.</td>
</tr>
</tbody>
</table>

**ALTER USER**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>ALTER USER [name] ([newname])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;alter-user name='...'&gt;newname='...'&lt;/alter-user&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>ADMIN</td>
</tr>
<tr>
<td>Summary</td>
<td>Renames the user with the specified name to newname.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified user does not exist, or if the new user already exists.</td>
</tr>
</tbody>
</table>

**ALTER PASSWORD**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>ALTER PASSWORD [name] ([password])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;alter-password name='...'&gt;([password])&lt;/alter-password&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>ADMIN</td>
</tr>
<tr>
<td>Summary</td>
<td>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).</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified user does not exist.</td>
</tr>
</tbody>
</table>
**DROP USER**

| Syntax           | DROP USER [name] (ON [pattern]):
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;drop-user name='...' (pattern='...')/&gt;</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>ADMIN</td>
</tr>
<tr>
<td>Summary</td>
<td>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 assigned database pattern will be removed.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if admin is specified as username, or if the specified user does not exist or is currently logged in.</td>
</tr>
</tbody>
</table>

**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 username 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**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>PASSWORD ([password])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;password&gt;([password])&lt;/password&gt;</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>Changes the password of the current user. If the command is run on command-line or in the GUI, the password can be omitted and entered interactively.</td>
</tr>
</tbody>
</table>

**Administration**

**SHOW OPTIONS**

*Updated with Version 10: Renamed (before: GET).*

<table>
<thead>
<tr>
<th>Syntax</th>
<th>SHOW OPTIONS [name]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;show-options (name='...')/</code>,</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>Returns the current values of all Options, or a single option with the specified name. Global options can only be requested by users with ADMIN permissions.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified option is unknown.</td>
</tr>
</tbody>
</table>

**SHOW SESSIONS**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>SHOW SESSIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Commands

<table>
<thead>
<tr>
<th><strong>XML Syntax</strong></th>
<th><code>&lt;show-sessions/&gt;</code>,</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Permission</strong></td>
<td><em>ADMIN</em></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows all sessions that are connected to the current server instance.</td>
</tr>
</tbody>
</table>

#### SHOW USERS

<table>
<thead>
<tr>
<th><strong>Syntax</strong></th>
<th><code>SHOW USERS (ON [database])</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Syntax</strong></td>
<td><code>&lt;show-users (database='...')/</code>,</td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td><em>ADMIN</em></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows all users that are visible to the current user. If a database is specified, only those users will be shown for which a pattern was specified that matches the database name.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>The command fails if the optional database could not be opened.</td>
</tr>
</tbody>
</table>

#### KILL

<table>
<thead>
<tr>
<th><strong>Syntax</strong></th>
<th><code>KILL [target]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Syntax</strong></td>
<td><code>&lt;kill target='...'/&gt;</code>,</td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td><em>ADMIN</em></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>The command fails if a user tried to kill his/her own session.</td>
</tr>
</tbody>
</table>

#### INFO DB

<table>
<thead>
<tr>
<th><strong>Syntax</strong></th>
<th><code>INFO DB</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Syntax</strong></td>
<td><code>&lt;info-db/&gt;</code>,</td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td><em>READ</em></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows general information and metadata on the currently opened database.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>The command fails if no database is opened.</td>
</tr>
</tbody>
</table>

#### INFO INDEX

| **Syntax** | `INFO INDEX ([ELEMNAME | ATTRNAME | PATH | TEXT | ATTRIBUTE | TOKEN | FULLTEXT])` |
|-------------|---------------------|
| **XML Syntax** | `<info-index type='elemname|attrname|path|text|attribute|token|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

<table>
<thead>
<tr>
<th><strong>Syntax</strong></th>
<th><code>INFO STORAGE [start end]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XML Syntax</strong></td>
<td><code>&lt;info-storage (start='...') (end='...')/</code>,</td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td><em>READ</em></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows the internal main table of the currently opened database. An integer range may be specified as argument.</td>
</tr>
</tbody>
</table>
## Commands

**Errors**

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

### 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 contents will be evaluated as command script. This command can be used to run several commands in a row, with no other transaction intervening the execution.

**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 evaluate the specified file as XQuery expression
- RUN commands.bxs, will evaluate 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 row, with no other transaction intervening the execution.

**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 "<commands><create-db name='db1'/><create-db name='db2'/></commands>", Two databases will be created in a single transaction.

#### 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])

---

65
Commands

**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 10

- Added: Backups: Support for general data (registered users, scheduled services and key-value stores).
- Added: DIR, GET
- Updated: SHOW OPTIONS: Renamed (before: GET).
- Updated: BINARY GET: Renamed (before: RETRIEVE).
- Updated: BINARY PUT: Renamed (before: STORE).
- Updated: PUT: Renamed (before: REPLACE).
- Updated: CREATE BACKUP: Support for comments.
- Updated: OPEN: path argument dropped.
- Removed: JOBS LIST, JOBS RESULT, JOBS STOP

Version 9.7

- Updated: ALTER DB, COPY: Overwrite existing databases.

Version 9.3

- Added: ALTER BACKUP

Version 8.6

- Updated: SHOW USERS: If called by non-admins, will only return the current user

Version 8.5

- Added: JOBS LIST, JOBS RESULT, JOBS STOP
• Updated: **Valid Names**: allow dots (except as first and last character)

**Version 8.4**

• Updated: `CREATE INDEX`, `DROP INDEX`, `INFO INDEX`: token index added

• Updated: `INFO STORAGE`: Query argument removed, start/end added to XML syntax.

• Updated: `INFO INDEX`: Token index added; index `TAG` renamed to `ELEMNAME`; index `ATTNAME` renamed to `ATTRNAME`

• Updated: `OPTIMIZE`: adopt original index options

**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`).
Commands

• 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 15. Options

The options listed on this page influence the way how database commands are executed and XQuery expressions are evaluated. Two kinds of options exist:

- **Global Options** are valid for all BaseX instances in the same JVM. This is particularly relevant if you are working with the client/server architecture.

- **Local options** (all remaining ones) are specific to a client or session.

Values of options are either strings, numbers or booleans. Options are static and not bound to a single operation (for example, the next command). Various ways exist to access and change options:

- The current value of an option can be requested with the `SHOW OPTIONS` command. Local options can be changed via `SET` (all global options, except for `DEBUG`, can only be changed at startup time). If an option is of type `boolean`, and if no value is specified, its current value will be inverted.

- The `.basex` configuration file is parsed by every new local BaseX instance. It contains all global options. Local options can be specified at the end of the file after the `Local Options` comment:

```
# General Options
DEBUG = false
...

# Local Options
CATALOG = etc/w3-catalog.xml
```

- 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.CATALOG=etc/w3-catalog.xml -cp basex.jar org.basex.BaseX -c"SHOW OPTIONS catalog"
CATALOG: etc/w3-catalog.xml
```

- If the Mac OS X packaged application is used, global options can be set within the Info.plist file within the Contents folder of the application package. For example:

```
<key>JVMOptions</key>
<array>
  <string>-Dorg.basex.CATALOG=etc/w3-catalog.xml</string>
</array>
```

- In a Web Application, the default can be adjusted in the `web.xml` file as follows:

```
<context-param>
  <param-name>org.basex.catalog</param-name>
  <param-value>etc/w3-catalog.xml</param-value>
</context-param>
```

- In XQuery, local options can be set via option declarations and pragmas.

If options are 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 Options

### DEBUG

<table>
<thead>
<tr>
<th>Signature</th>
<th>DEBUG [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>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 <code>-d</code>.</td>
</tr>
</tbody>
</table>

### DBPATH

<table>
<thead>
<tr>
<th>Signature</th>
<th>DBPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>{home}/data</td>
</tr>
<tr>
<td>Summary</td>
<td>Points to the directory in which all databases are located.</td>
</tr>
</tbody>
</table>

### LOGPATH

<table>
<thead>
<tr>
<th>Signature</th>
<th>LOGPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>.logs</td>
</tr>
<tr>
<td>Summary</td>
<td>Points to the directory in which all log files are stored. Relative paths will be resolved against the DBPATH directory.</td>
</tr>
</tbody>
</table>

### REPOPATH

<table>
<thead>
<tr>
<th>Signature</th>
<th>REPOPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>{home}/repo</td>
</tr>
<tr>
<td>Summary</td>
<td>Points to the Repository, in which all XQuery modules are located.</td>
</tr>
</tbody>
</table>

### LANG

<table>
<thead>
<tr>
<th>Signature</th>
<th>LANG [language]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>English</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies the interface language. Currently, seven languages are available: 'English', 'German', 'French', 'Dutch', 'Italian', 'Japanese', and 'Vietnamese'.</td>
</tr>
</tbody>
</table>

### LANGKEY

<table>
<thead>
<tr>
<th>Signature</th>
<th>LANGKEY [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

### FAIRLOCK

<table>
<thead>
<tr>
<th>Signature</th>
<th>FAIRLOCK [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>Defines the locking strategy:</td>
</tr>
<tr>
<td></td>
<td>• By default, non-fair is used. Read transactions will be favored, and transactions that access no databases can be evaluated even if the limit of parallel transactions (specified via PARALLEL) has been reached. This prevents update operations from blocking all other requests. For example, the DBA can further be used to see which jobs are running, even if the queue is full.</td>
</tr>
</tbody>
</table>
• If fair locking is enabled, read and write transactions will be treated equally (first in, first out). This avoids starvation of update operations, and it should be used if the prompt evaluation of update operations is critical.

**CACHETIMEOUT**

<table>
<thead>
<tr>
<th>Signature</th>
<th>CACHETIMEOUT [seconds]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>3600</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies how many seconds the results of queries, which have been queued by the asynchronously executed, will be cached in main memory.</td>
</tr>
</tbody>
</table>

**Client/Server Architecture**

**HOST**

<table>
<thead>
<tr>
<th>Signature</th>
<th>HOST [host]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>localhost</td>
</tr>
<tr>
<td>Summary</td>
<td>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 <code>-n</code>.</td>
</tr>
</tbody>
</table>

**PORT**

<table>
<thead>
<tr>
<th>Signature</th>
<th>PORT [port]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1984</td>
</tr>
<tr>
<td>Summary</td>
<td>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 <code>-p</code>.</td>
</tr>
</tbody>
</table>

**SERVERPORT**

<table>
<thead>
<tr>
<th>Signature</th>
<th>SERVERPORT [port]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1984</td>
</tr>
<tr>
<td>Summary</td>
<td>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 <code>-p</code>.</td>
</tr>
</tbody>
</table>

**USER**

<table>
<thead>
<tr>
<th>Signature</th>
<th>USER [name]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>Represents a username, which is used for accessing the server or an HTTP service:</td>
</tr>
<tr>
<td></td>
<td>• The default value will be overwritten if a client specifies its own credentials.</td>
</tr>
<tr>
<td></td>
<td>• If the default value is empty, login will only be possible if the client specifies credentials.</td>
</tr>
<tr>
<td></td>
<td>• The option can also be changed on command line via <code>-U</code>.</td>
</tr>
</tbody>
</table>

**PASSWORD**

<table>
<thead>
<tr>
<th>Signature</th>
<th>PASSWORD [password]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>Represents a password, which is used for accessing the server:</td>
</tr>
<tr>
<td></td>
<td>• The default value will be overwritten if a client specifies its own credentials.</td>
</tr>
</tbody>
</table>
Options

- If the default value is empty, authentication will only be possible if the client supplies 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**

<table>
<thead>
<tr>
<th>Signature</th>
<th>AUTHMETHOD [method]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Basic</td>
</tr>
</tbody>
</table>
| Summary      | Specifies the default authentication method, which will be used by the HTTP server for negotiating credentials. Allowed values are Basic, Digest, and Custom:  
  - If basic access is chosen, the client can still request digest authentication.  
  - This is different for digest access, which cannot be overwritten.  
  - With custom authentication, the server will not do any authentication. |

**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**

<table>
<thead>
<tr>
<th>Signature</th>
<th>PROXYHOST [host]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>This is the host name of a proxy server. If the value is an empty string, it will be ignored.</td>
</tr>
</tbody>
</table>

**PROXYPORT**

<table>
<thead>
<tr>
<th>Signature</th>
<th>PROXYPORT [port]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Summary</td>
<td>This is the port number of a proxy server. If the value is set to 0, it will be ignored.</td>
</tr>
</tbody>
</table>

**NONPROXYHOSTS**

<table>
<thead>
<tr>
<th>Signature</th>
<th>NONPROXYHOSTS [hosts]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>This is a list of hosts that should be directly accessed. If the value is an empty string, it will be ignored.</td>
</tr>
</tbody>
</table>

**IGNORECERT**

*Updated with Version 10:* Additionally disable hostname verification.

<table>
<thead>
<tr>
<th>Signature</th>
<th>IGNORECERT [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>Disable SSL hostname verification and ignore untrusted certificates when connecting to servers.</td>
</tr>
</tbody>
</table>
## OPTIONS

### TIMEOUT

<table>
<thead>
<tr>
<th>Signature</th>
<th>TIMEOUT [seconds]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>30</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies the maximum time a transaction triggered by a client may take. If an operation takes longer than the specified number of seconds, it will be aborted. Active update 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 operations with admin permissions.</td>
</tr>
</tbody>
</table>

### KEEPALIVE

<table>
<thead>
<tr>
<th>Signature</th>
<th>KEEPALIVE [seconds]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>600</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

### PARALLEL

<table>
<thead>
<tr>
<th>Signature</th>
<th>PARALLEL [number]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>8</td>
</tr>
<tr>
<td>Summary</td>
<td>Denotes the maximum allowed number of parallel transactions:</td>
</tr>
<tr>
<td></td>
<td>• If FAIRLOCK is enabled, the number of parallel transactions will never exceed the specified value.</td>
</tr>
<tr>
<td></td>
<td>• If the option is disabled (which is the default), the limit only applies to transactions that access databases.</td>
</tr>
<tr>
<td></td>
<td>• The main reason for allowing parallel operations is to prevent slow transactions from blocking all other operations. 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.</td>
</tr>
</tbody>
</table>

### LOG

<table>
<thead>
<tr>
<th>Signature</th>
<th>LOG [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

### LOGMSGMAXLEN

<table>
<thead>
<tr>
<th>Signature</th>
<th>LOGMSGMAXLEN [length]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1000</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies the maximum length of a single log message.</td>
</tr>
</tbody>
</table>

### LOGTRACE

<table>
<thead>
<tr>
<th>Signature</th>
<th>LOGTRACE [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>
### Options

#### Summary
If BaseX is running as Web Application, trace output (generated via `fn:trace`, `prof:dump` and similar functions) is written to the database logs. If this option is disabled, trace output will be redirected to standard error, as it is known from the standalone version of BaseX.

#### HTTP Services

Most HTTP options are defined in the `jetty.xml` and `web.xml` configuration files in the `webapp/WEB-INF` directory. Some additional BaseX-specific options exist that will be set before the web server is started:

### WEBPATH

**Signature**

<table>
<thead>
<tr>
<th>WEBPATH [path]</th>
</tr>
</thead>
</table>

**Default**

(home)/webapp

**Summary**

Points to the directory in which all the Web Application contents are stored, including XQuery, Script, RESTXQ and configuration files:
- The option is ignored if BaseX is deployed as web servlet.
- It cannot be assigned via the `web.xml` file, as it will be evaluated before the configuration files are parsed.

### GZIP

**Signature**

<table>
<thead>
<tr>
<th>GZIP [boolean]</th>
</tr>
</thead>
</table>

**Default**

false

**Summary**

Jetty provides a Gzip handler for dynamically uncompressing requests and compressing responses. This feature can be enabled if Jetty is started via the BaseX HTTP Server:
- The option can also be enabled on command line via `-g`.
- It cannot be assigned via the `web.xml` file, as it will be evaluated before the configuration files are parsed.
- In addition to the Jetty defaults (GET requests, exclusion of binaries, MSIE 6.0, etc.), POST and PUT requests are supported.

### RESTXQPATH

**Signature**

<table>
<thead>
<tr>
<th>RESTXQPATH [path]</th>
</tr>
</thead>
</table>

**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.

### PARSERESTXQ

**Signature**

<table>
<thead>
<tr>
<th>PARSERESTXQ</th>
</tr>
</thead>
</table>

**Default**

3

**Summary**

Timeout after which the RESTXQ directory will be parsed for changes:
- If 0 is specified, the directory will be parsed every time a RESTXQ function is called.
- A positive value defines the idle time in seconds after which parsing will be enforced. The default value is 3: Changes in the RESTXQ directory will be detected after 3 seconds without RESTXQ function calls.
- Monitoring is completely disabled if a negative value is specified.
Options

RESTXQERRORS

Signature    RESTXQERRORS
Default      true
Summary      Reports parsing errors in XQuery modules in the RESTXQ directory and returns the full error message and stack trace to the client. By default, this option is enabled. In a production environment, it can be disabled to suppress errors that should not be seen by the user of an API (the full error information can still be looked up in the database logs). See RESTXQ Error Handling for more details.

RESPATH

Signature    RESPATH [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, the database server instance will be started in addition, which can then be addressed by Clients via the database port (see PORT). If the option is set to true, no database server will be launched.

STOPPORT

Signature    STOPPORT [port]
Default      8081
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 created in main memory:

• Most queries will be evaluated faster in main-memory mode, but all data is lost if the BaseX instance in which the database was created is shut down.
• It is not possible to store binary resources in a main-memory database.
A main-memory database will have no disk representation. However, it is possible to export the database via the `export` command, and create a new database from the exported file in a second step.

This option will not be available for `db:create`, because the database would not be accessible anymore after database creation, i.e., outside the query scope.

**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.

**ARCHIVENAME**

**Signature** 
ARCHIVENAME [boolean]

**Default** 
false

**Summary** 
If this option is set to true, the file name of parsed archives will be included in the document paths.

**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 enabled, all resources that are filtered out by the CREATEFILTER option while being added to a database will be stored as raw files instead (i.e., in their binary representation).
### PARSER

**Signature**  
PARSER [type]

**Default**  
XML

**Summary**  
Defines a parser for importing new files to the database. Available parsers are XML, JSON, CSV, TEXT, HTML, and RAW. HTML input will be parsed as XML documents if Tagsoup is not found in the classpath.

### CSV_PARSER

**Signature**  
CSV_PARSER [options]

**Default**  
empty

**Summary**  
Specifies the way how CSV data will be parsed. Keys and values are delimited with =, and multiple options are delimited with ,. The available options (except for the additional encoding option) are described in the CSV Module.

**Examples**  
encoding=CP1252,header=true parses the input as CP1252 and the first line as header.

### JSON_PARSER

**Signature**  
JSON_PARSER [options]

**Default**  
empty

**Summary**  
Specifies the way how JSON data will be parsed. Keys and values are delimited with =, and multiple options are delimited with ,. The available options (except for the additional encoding option) are described in the JSON Module.

**Examples**  
format=jsonml,lax=yes interprets the input as JSONML and uses lax parsing.

### HTML_PARSER

**Signature**  
HTML_PARSER [options]

**Default**  
empty

**Summary**  
Specifies the way how HTML data will be parsed. Keys and values are delimited with =, and multiple options are delimited with ,. The available options are described in the Parsers article.

**Examples**  
- encoding=Shift-JIS,nons=true parses the input as Shift-JIS and suppresses namespaces.
- lexical=true preserves comments.

### TEXT_PARSER

**Signature**  
TEXT_PARSER [options]

**Default**  
empty

**Summary**  
Specifies the way how TEXT data will be parsed. Keys and values are delimited with =, and multiple options are delimited with ,. The available options are listed in the Parsers article.

**Examples**  
lines=true creates a single element for each line of text.

### XML Parsing

### STRIPWS


**Signature**  
STRIPWS [boolean]
Many XML documents include whitespaces that have been added to improve readability. This option controls the white-space processing mode of the XML parser:

- If the option is set to true, leading and trailing whitespaces from text nodes will be stripped, and empty text nodes will be discarded.
- The flag should be disabled if a document contains mixed content.
- The flag can also be turned on via the command line and -w.
- If the option is enabled, whitespaces of an element and its descendants can locally be preserved with the `xml:space="preserve"` attribute:

```xml
<xml>
  <title>
    Demonstrating the STRIPWS flag
  </title>
  <text xml:space="preserve">To <b>be</b>, or not to <b>be</b>, that is the question.</text>
</xml>
```

If whitespaces are stripped, `indent=yes` can be assigned to the SERIALIZE option to get properly indented XML output.

**STRIPNS**

- **Signature**: STRIPNS [boolean]
- **Default**: false
- **Summary**: Strips all namespaces from an XML document while parsing.

**INTPARSE**

- **Signature**: INTPARSE [boolean]
- **Default**: false
- **Summary**: Uses the internal XML parser instead of the standard Java XML parser. Here are some reasons for using the internal parser:
  - Performance: Documents (in particular small ones) will be parsed faster
  - Fault tolerance: invalid characters will automatically be replaced with the Unicode replacement character `FFFD (#)`
  - Entities: around 250 HTML entities will be detected and decoded

You will be able to correctly parse most XML documents with the internal parser. Java’s Xerces parser is still used as default, however, because it supports all features of the XML standard and advanced DTD features, such as recursive entity expansion.

**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 CATALOG option can be changed to locally resolve DTDs.
Options

### 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 INTFPARSE).

### CATALOG

*Updated with Version 10:* Renamed from CATFILE.

**Signature** | CATALOG [path]
---|---
**Default** | empty

**Summary**
Semicolon-separated list of XML catalog files to resolve URIs. See Catalog Resolvers for more details.

### Indexing

The following options control the creation of index structures. The current values will be considered if a new database is created. See Indexes for more details.

#### 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 Text Index 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 Attribute Index for more details.

#### TOKENINDEX

**Signature** | TOKENINDEX [boolean]
---|---
**Default** | true

**Summary**
Creates a token index whenever a new database is created. A token index speeds up searches for single tokens in attribute values. See Token Index 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 Full-Text Index for more details.

### TEXTINCLUDE

**Signature** | TEXTINCLUDE [names]
---|---
## Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Signature</th>
<th>Default</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>empty</td>
<td><strong>Summary</strong></td>
<td>Defines name patterns for the parent elements of texts that are indexed. By default, all text nodes will be indexed. Name patterns are separated by commas. See Selective Indexing for more details.</td>
</tr>
<tr>
<td><strong>ATTRINCLUDE</strong></td>
<td>ATTRINCLUDE [names]</td>
<td>empty</td>
<td>Defines name patterns for the attributes to be indexed. By default, all attribute nodes will be indexed. Name patterns are separated by commas. See Selective Indexing for more details.</td>
</tr>
<tr>
<td><strong>TOKENINCLUDE</strong></td>
<td>TOKENINCLUDE [names]</td>
<td>empty</td>
<td>Defines name patterns for the attributes to be indexed. By default, tokens in all attribute nodes will be indexed. Name patterns are separated by commas. See Selective Indexing for more details.</td>
</tr>
<tr>
<td><strong>FTINCLUDE</strong></td>
<td>FTINCLUDE [names]</td>
<td>empty</td>
<td>Defines name patterns for the parent elements of texts that are indexed. By default, all text nodes will be indexed. Name patterns are separated by commas. See Selective Indexing for more details.</td>
</tr>
<tr>
<td><strong>MAXLEN</strong></td>
<td>MAXLEN [int]</td>
<td>96</td>
<td>Specifies the maximum length for strings to be stored in index structures. The value of this option will be assigned once to a new database, and can only be changed by creating a new database or doing a full optimization.</td>
</tr>
<tr>
<td><strong>MAXCATS</strong></td>
<td>MAXCATS [int]</td>
<td>100</td>
<td>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.</td>
</tr>
<tr>
<td><strong>UPDINDEX</strong></td>
<td>UPDINDEX [boolean]</td>
<td>false</td>
<td>If turned on, incremental indexing will be enabled:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• The current value of this option will be assigned to new databases. It can be changed for existing databases by running OPTIMIZE with the ALL keyword or db:optimize and true() as second argument.</td>
</tr>
</tbody>
</table>
Options

- After each update, the value indexes will be refreshed as well. Incremental updates are currently not available for the full-text index and database statistics.
- Find more details in the article on Index Structures.

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.

SPLITSIZE

Signature | SPLITSIZE [num]
---|---
Default | 0
Summary | This option affects the construction of new value indexes. It controls 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 heuristics are applied, based on the current memory consumption. Usually, this works fine.
- If explicit garbage collection is disabled when running Java (e.g. via the JVM option -XX:+DisableExplicitGC), you may need to choose a custom split size.
- You can e.g. start with 1000000 (one million) index operations and adjust this value in the next steps.
- The larger the assigned value is, the less splits will take place, and the more main memory will be required.

Full-Text Indexing

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, tokens will not be stemmed. See Full-Text Index for more details.

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). See Full-Text Index for more details.
Options

<table>
<thead>
<tr>
<th>DIACRITICS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>DIACRITICS [boolean]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>If set to true, diacritics will be preserved during full-text indexing. By default, diacritics will be removed. See Full-Text Index for more details.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LANGUAGE</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>LANGUAGE [lang]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>en</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>The specified language will influence the way how texts will be tokenized and stemmed. It can be the name of a language or a language code. See Full-Text Index for more details.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STOPWORDS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>STOPWORDS [path]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>If a text file with stop words is specified, frequently used terms contained in that file will be ignored when a full-text index is created. A stopword list may decrease the size of the full text index and speed up your queries. See Full-Text Index for more details.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query Options

<table>
<thead>
<tr>
<th>QUERYINFO</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>QUERYINFO [boolean]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>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.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MIXUPDATES</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>MIXUPDATES</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>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, in compliance with the XQuery Update Facility, this option is set to false. See Returning Results for more details.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BINDINGS</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td>BINDINGS [vars]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>empty</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Contains external variables to be bound to a query. The string must comply with the following rules:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Variable names and values must be separated by equality signs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Multiple variables must be delimited by commas.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Commas in values must be duplicated.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Variables may optionally be introduced with a leading dollar sign.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• 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 $GREETING$.

```xquery
SET BINDINGS GREETING="hello world!"
XQUERY declare variable $GREETING external; $GREETING
```

**INLINELIMIT**

**Signature**

INLINELIMIT

**Default**

50

**Summary**

This option controls inlining of XQuery functions:

- 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 overwritten via the %basex:inline annotation (follow the link to get more information on function inlining).

**UNROLLLLIMIT**

**Signature**

UNROLLLLIMIT

**Default**

5

**Summary**

This option controls the unroll limit:

- Loops with few iterations are unrolled by the XQuery compiler to enable further optimizations.
- If the limit is increased, more optimizations will take place, but the memory consumption and compile time will increase.
- See Loop Unrolling for more details.

**ENFORCEINDEX**

**Signature**

ENFORCEINDEX [boolean]

**Default**

false

**Summary**

Enforces index rewritings in path expressions. See Enforce Rewritings for details.

**COPYNODE**

**Signature**

COPYNODE [boolean]
Options

<table>
<thead>
<tr>
<th>Default</th>
<th>true</th>
</tr>
</thead>
</table>
| Summary | When creating new nodes in XQuery via Node Constructors, copies of all enclosed nodes will be created, and the copied nodes get new node identities. As a result, the following query yields false:

```xquery
let $a := <a/>
let $b := <b>{ $a }</b>
return $b/a is $a
```

This step can be very expensive and memory consuming. If the option is disabled, child nodes will only be linked to the new parent nodes, and the upper query returns true. The option should be used carefully as it changes the semantics of XQuery. It should preferably be used in Pragmas.

TAILCALLS

<table>
<thead>
<tr>
<th>Signature</th>
<th>TAILCALLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>256</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

WITHDB

<table>
<thead>
<tr>
<th>Signature</th>
<th>WITHDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>
| Summary   | By default, resources specified via fn:doc and fn:collection are looked up both in the database and in the file system. If you always use db:get to access databases, it is recommendable to disable this option:

- No locks will be created for the two functions (see limitations of database locking for more details).
- Access to local and external resources will be faster, as the database lookup will be skipped. |

DEFAULTDB

<table>
<thead>
<tr>
<th>Signature</th>
<th>DEFAULTDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>If this option is turned on, paths specified in the fn:doc and fn:collection 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.</td>
</tr>
</tbody>
</table>

FORCECREATE

<table>
<thead>
<tr>
<th>Signature</th>
<th>FORCECREATE [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>By activating this option, database instances will be created with the XQuery functions fn:doc and fn:collection.</td>
</tr>
</tbody>
</table>

CHECKSTRINGS

<table>
<thead>
<tr>
<th>Signature</th>
<th>CHECKSTRINGS [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>
By default, characters from external sources that are invalid in XML will trigger an error. If the option is set to false, these characters will be replaced with the Unicode replacement character FFFD (#). The option affects Java Bindings and string conversion and input functions such as archive:create, archive:extract-text, and archive:update.

**WRAPJAVA**

<table>
<thead>
<tr>
<th>Signature</th>
<th>WRAPJAVA [mode]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>some</td>
</tr>
<tr>
<td>Summary</td>
<td>Defines the way how values that result from Java code invocation will be converted to XQuery items. Allowed values: none, all, some, instance, and void. See Java Bindings for further details.</td>
</tr>
</tbody>
</table>

**LSERROR**

<table>
<thead>
<tr>
<th>Signature</th>
<th>LSERROR [error]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0</td>
</tr>
<tr>
<td>Summary</td>
<td>This option specifies the maximum Levenshtein error for fuzzy full-text matching. By default, if 0 is assigned, the error value is calculated dynamically. See Fuzzy Querying for more details.</td>
</tr>
</tbody>
</table>

**RUNQUERY**

<table>
<thead>
<tr>
<th>Signature</th>
<th>RUNQUERY [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies if a query will be executed or parsed only. This option can also be changed on command line via -R.</td>
</tr>
</tbody>
</table>

**RUNS**

<table>
<thead>
<tr>
<th>Signature</th>
<th>RUNS [num]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Serialization Options**

**SERIALIZE**

<table>
<thead>
<tr>
<th>Signature</th>
<th>SERIALIZE [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

**SERIALIZER**

<table>
<thead>
<tr>
<th>Signature</th>
<th>SERIALIZER [params]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>Parameters for serializing query results:</td>
</tr>
<tr>
<td></td>
<td>- Variable names and values are separated by equality signs.</td>
</tr>
</tbody>
</table>
Options

• Multiple variables are delimited by commas.
• Commas must be duplicated if they appear as literals in values.

The option can also be used on command line with the flag -s.

Examples

• indent=yes : enables automatic indentation of XML nodes. This is recommended if whitespaces have been stripped from a document (see Template:Options).
• 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 resources of a database and writing files after updates via the WRITEBACK option. Keys and values are separated by equality signs, multiple parameters are delimited by commas. See Serialization for more details.

Examples

• indent=no,omit-xml-declaration=no : disables automatic indentation of XML nodes, outputs the XML declaration.

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.

FULLPLAN

Signature FULLPLAN [boolean]
Default false
Summary Attaches the file path, line and column of the expressions in the original query string to the query plan. Values (items and sequences) have no input information attached.

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 and fn:collection back to disk.
Options

- This option can also be activated on command line via `-u`.
- Please take in mind that no backup will be created from your original files.
- The serialization options can be controlled via the `EXPORTER` option.

MAXSTAT

<table>
<thead>
<tr>
<th>Signature</th>
<th>MAXSTAT [num]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>30</td>
</tr>
</tbody>
</table>

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

Changelog

Version 10.0
- Removed: `COMPPLAN`, `IGNOREHOSTNAME`
- Updated: `IGNORECERT`: Additionally disable hostname verification.
- Updated: `CATALOG`: Renamed (before: `CATFILE`).
- Updated: `STRIPWS`: Renamed (before: `CHOP`), new default: false.

Version 9.7
- Updated: `GZIP`: Support for POST and PUT requests.

Version 9.6
- Added: `UNROLLLIMIT, WRAPJAVA`

Version 9.5
- Updated: `INLINELIMIT`: default reduced to 50.
- Updated: `RESTXQERRORS`: additionally suppress stack trace in HTTP response

Version 9.4
- Added: `LOGTRACE`

Version 9.3
- Added: `WITHDB, GZIP`

Version 9.2
- Added: `RESTXQERRORS, FULLPLAN`
- Removed: `DOTPLAN, DOTCOMPACT`

Version 9.0
- Added: `ENFORCEINDEX, COPYNODE, IGNOREHOSTNAME`

Version 8.6
- Added: `FAIRLOCK, PARSERESTXQ`
- Removed: `GLOBALLOCK` (exclusive use of database lock)
Options

- Removed: QUERYPATH (will now be internally assigned)
- Removed: CACHERESTXQ (replaced with PARSERESTXQ)

Version 8.5
- Added: CACHETIMEOUT, LOGPATH
- Updated: AUTHMETHOD: custom value added.

Version 8.4
- Added: TOKENINDEX, TOKENINCLUDE
- Added: SPLITSIZE (replacing INDEXSPLITSIZE and FTINDEXSPLITSIZE)
- Removed: INDEXSPLITSIZE, FTINDEXSPLITSIZE

Version 8.3
- Added: CACHERESTXQ, TEXTINCLUDE, ATTRINCLUDE, FTINCLUDE, ARCHIVENAME

Version 8.2
- Removed: EVENTPORT, CACHEQUERY

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: ADCACHE, CHECKSTRINGS, FTINDEXSPLITSIZE, INDEXSPLITSIZE

Version 7.6
- Added: GLOBALLOCK
- Added: store local options in configuration file after # Local Options comments.
Options

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, RESTXQPATHHTTPLOCAL, 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, PROXYPORy, 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
Chapter 16. Configuration

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. It gives some 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.

An empty `.basexhome` file can be added to a directory to mark it as home directory.

Note that:

- Depending on your OS and configuration, files and folders with a `.` prefix may be hidden.
- In the Web Application context, options can be defined in the `web.xml` file.

**Home Directory**

As BaseX is distributed in different flavors, and as it may be started from different locations, it dynamically determines its home directory:

- First, the **Java 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 selected if the `.basex` or `.basexhome` file is found in this directory.
- If not, the **application directory** (the folder in which BaseX is located) is chosen if one of these two files is found in that directory.
- In all other cases, a `basex` subdirectory in the **user home directory** will be returned. The user home directory is retrieved via the `HOME` environment variable, or (if unassigned) the Java system property `user.home`.

If BaseX is used in an embedded environment (such as a servlet in a Web Application), it may not immediately be clear which directory was picked. You can run the XQuery expression `Q{java:org.basex.util.Prop}HOMEDIR()` to find out.

**Database Directory**

Databases consists of several binary files. These are located in a directory named by the name of the database. The database root directory is 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 automatically be moved to the new destination.

**Log Files**

Log files are stored in text format in a `.logs` subdirectory of the database folder (see Logging for more information).
Configuration

Changelog

Version 9.0

• Updated: Detection and configuration of home directory and subdirectories.

Version 8.0

• Updated: .basexpem 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.
Part V. 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 alternatives how to use BaseX in oXygen:

- Resources in BaseX databases can be opened and modified.
- XPath/XQuery 1.0 expressions can be run by the query processor of BaseX.

**Note:** BaseX itself is a highly compliant XQuery 3.1 processor. The restriction to XQuery 1.0 arises from the XQJ Interface which is used to establish the connection between oXygen and BaseX. We strongly encourage you to use the XML editor integrated into the BaseX GUI to edit and query your XML data!

Access Database Resources

Preparations

1. Download one of the BaseX distributions.
2. Start BaseX (see Startup).
3. Create a BaseX database, if necessary (see Databases).
4. Start the BaseX WebDAV service.

Configuration

1. In oXygen, go to menu Options → Preferences → Data Sources.
2. In the Connections panel (in the lower half of the screen), click the New button (+).
3. Enter "BaseX WebDAV" as connection name.
4. Select "WebDAV (S)FTP" in the Data Source dropdown box.
5. Fill in the appropriate connection details as follows:
   - Set the username to admin and enter your password.
6. Now press OK, and your Data Source is ready for use.

You can then access your database file(s) via the Data Source Explorer: Window → Show View → Data Source Explorer.
Integrating oXygen

Perform Queries

One-Time Setup

Preparations

1. Download one of the complete BaseX distributions (ZIP, EXE), if necessary.

2. Start BaseX (see Startup). Note: 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.

Configure Data Source

1. In oXygen, select Options → Preferences → Data Sources.

2. In the Data Sources panel, add a new data source using the New button (+).

3. Enter "BaseX" as name and select XQuery API for Java(XQJ) from the Type dropdown box.

4. Add the following JAR files (downloaded in Preparations procedure) with the Add Files Button. The versions of the JAR files may differ.
   - basex/lib/xqj-api-1.0.jar
   - basex/lib/xqj2-0.2.0.jar
   - basex/lib/basex-xqj-9.0.jar
   - basex/BaseX.jar, if you want to use BaseX embedded

5. Under "Driver class", choose the preferred driver class:
   - Client/server communication: net.xqj.basex.BaseXXQDataSource
   - Embedded use (standalone): net.xqj.basex.local.BaseXXQDataSource

6. Click OK.

Configure Connection

1. In the Connections section (in the lower half of the Data Source dialog), click New (+).

2. Enter "BaseX XQJ" as name and select "BaseX" as data source.

3. If you use the default driver, enter the following values in the Connection Details section:
   - port: 1984
   - serverName: localhost
   - user: admin
   - password: your password

4. Click OK to complete the connection configuration.

5. Click OK again to close the Preferences dialog.

Configure New Transformation Scenario

1. Select Window → Show View → Transformation Scenarios.
Integrating oXygen

2. In the Transformation Scenarios panel, click + and select XQuery transformation in the lower part of the dropdown list.

3. Enter a name for your transformation, e.g. "BaseX" like in the screenshot below.

4. Specify an optional XML and XQuery URL.
   • If you would like to query the BaseX database you connected to via WebDAV, leave the XML URL field empty. To access your database, you can use the following function from the BaseX Database Module in your XQuery URL file:
   • If you specify an XML document in the XML URL field, you can query its content using . (dot operator) in your XQuery URL file.

5. Choose "BaseX XQJ" as Transformer from the combo box.

6. Click OK to complete the scenario configuration.

![Transformation Scenarios - x.xq](image)

**Execute Query**

After the one-time setup steps are complete, you can execute your query using the new transformation scenario. Start the transformation by clicking the red Run button (Apply associated scenarios) in the Transformation Scenarios window, while your scenario is selected. The results should be immediately displayed in the result panel.
Chapter 18. Integrating Eclipse

Read this entry online in the BaseX Wiki.

This tutorial is part of the Getting Started Section. It describes how to access BaseX from Eclipse via the oXygen XML Editor plugin. The plugin offers the same features as specified in Integrating oXygen. However, the way to get there from within Eclipse is a little bit different.

Currently, there are two alternatives how to use BaseX in oXygen:

- Resources in BaseX databases can be opened and modified.
- XPath/XQuery 1.0 expressions can be run by the query processor of BaseX.

**Note:** BaseX itself is a highly compliant XQuery 3.1 processor. The restriction to XQuery 1.0 arises from the XQJ Interface which is used to establish the connection between oXygen and BaseX. We strongly encourage you to use the XML editor integrated into the BaseX GUI to edit and query your XML data!

**Preparations**

1. Download and install Eclipse. **Note:** The current version of the oXygen XML Editor plugin was tested for Eclipse Version 4.8. Please also note that you will require an oXygen license to use the plugin.

2. Follow the instructions in the oXygen Manual to install the plugin.

3. In Eclipse, click on the oXygen icon in the upper right corner to open the plugin. The XML Project you created during the installation of the plugin should be displayed in the Navigator panel. In this example, it is called `BaseXProject`.

**Access Database Resources**

**Preparations**

1. Download one of the BaseX distributions.

2. Start BaseX (see Startup).

3. Create a BaseX database, if necessary (see Databases).

4. Start the BaseX WebDAV service.

**Configuration**

**Note:** If you have already integrated BaseX into the oXygen XML Editor itself as described in Integrating oXygen, your BaseX WebDAV connection will already be available in the plugin.

1. In Eclipse, go to menu `Eclipse → Preferences`. In the `Preferences` dialog, chose the `oXygen XML Editor` item, and then the `Data Sources` subitem.

2. In the Connections panel (in the lower half of the screen), click the `New` button (+).

3. Enter "BaseX WebDAV" as connection name.

4. Select "WebDAV (S)FTP" in the Data Source dropdown box.

5. Fill in the appropriate connection details as follows:
   - Set the username to `admin` and enter your password.
6. Press OK to close the dialog.

7. Click Apply and Close to close the Preferences dialog.

8. If prompted, restart Eclipse to activate all changes.

You can then access your database file(s) via the Data Source Explorer: Windows → Show View → Data Source Explorer.

Perform Queries

One-Time Setup

Note: If you have already integrated BaseX into the oXygen XML Editor itself as described in Integrating oXygen, your data sources and connections will already be available in the plugin.

Configuration

1. In Eclipse, go to menu Eclipse → Preferences. In the Preferences dialog, chose the oXygen XML Editor item, and then the Data Sources subitem.

2. In the Data Sources panel, add a new data source using the New button (+).

3. Enter "BaseX" as name and select XQuery API for Java(XQJ) from the Type dropdown box.

4. Add the following JAR files (downloaded in Preparations procedure) with the Add Files Button. The versions of the JAR files may differ.
   • basex/lib/xqj-api-1.0.jar
   • basex/lib/xqj2-0.2.0.jar
   • basex/lib/basex-xqj-9.0.jar
   • basex/BaseX.jar, if you want to use BaseX embedded

5. Under "Driver class", choose the preferred driver class:
   • Client/server communication: net.xqj.basex.BaseXXQDataSource
   • Embedded use (standalone): net.xqj.basex.local.BaseXXQDataSource

6. Click OK.

Configure Connection

1. In the Connections section (in the lower half of the Data Source dialog), click New (+).
2. Enter "BaseX XQJ" as name and select "BaseX" as data source.

3. If you use the default driver, enter the following values in the Connection Details section:
   - port: 1984
   - serverName: localhost
   - user: admin
   - password: your password

4. Click OK to complete the connection configuration.

5. Click Apply and Close to close the Preferences dialog.

6. If prompted, restart Eclipse to activate all changes.

**Configure New Transformation Scenario**

1. In Eclipse, choose File → New → XQuery File. Enter a filename and click Finish. Enter a query and save the file.

2. Select Window → Show View → Transformation Scenarios.

3. In the Transformation Scenarios panel on the right-hand side, click + and select XQuery transformation in the lower part of the dropdown list.

4. Enter a name for your transformation, e.g. "BaseX".

5. Specify an optional XML and XQuery URL.
   - If you would like to query the BaseX database you connected to via WebDAV, leave the XML URL field empty. To access your database, you can use the db:get function from the BaseX Database Module in your XQuery URL file.
   - If you specify an XML document in the XML URL field, you can query its content using . (dot operator) in your XQuery URL file.

6. Choose "BaseX XQJ" as Transformer from the combo box.

7. Click OK to complete the scenario configuration.

**Execute Query**

After the one-time setup steps are complete, you can execute your query using the new transformation scenario. Start the transformation by clicking the red Run button (Apply associated scenarios) in the Transformation Scenarios window, while your scenario is selected. The results should be immediately displayed in the result panel.
Chapter 19. Integrating IntelliJ IDEA

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 IntelliJ IDEA IDE. There are currently two XQuery plugins for IntelliJ IDEA on the market:

- The xquery-intellij-plugin by Reece H. Dunn.
- The XQuery Support plugin by Grzegorz Ligas.
- Both plugins offer support for XQuery 3.1 and can be run as a client or standalone instance. Please note that the two plugins are mutually exclusive and cannot be activated at the same time in IntelliJ.

**Note:** BaseX itself is a highly compliant XQuery 3.1 processor. We strongly encourage you to use the XML editor integrated into the BaseX GUI to edit and query your XML data!

Preparations

The following steps apply to all operating systems and both plugins:

- Install either version of IntelliJ IDEA: the Community or Ultimate edition.
- Download your favorite BaseX distribution (JAR, ZIP, EXE).
- Start BaseX (see Startup).
- Create a BaseX database (see Databases).

**xquery-intellij-plugin**

This section focuses on Reece H. Dunn’s xquery-intellij-plugin.

Installation

After installing IntelliJ IDEA and BaseX, install the xquery-intellij-plugin by one of the following methods:

From the Start Screen

- Start IntelliJ IDEA and select **Configure → Plugins**.
- In the Plugins window, select the tab **Marketplace**.
- Type "XQuery" into the **Search plugins in marketplace** field.
Integrating IntelliJ IDEA

- Click the Install button below xquery-intellij-plugin.
- You will be prompted to restart IDEA to load the new plugin.

From the IntelliJ IDEA Menu

- Select Settings (Windows)/Preferences (macOS) in the IntelliJ IDEA menu.
- In the Settings/Preferences window, select Plugins.
- In the Plugins window, select the tab Marketplace.
- Type "XQuery" into the Search plugins in marketplace field.
- Click the Install button below xquery-intellij-plugin plugin.
- You will be prompted to restart IDEA to load the new plugin.

Configuring The Processor

- Start IntelliJ IDEA and navigate to Settings (Windows)/Preferences (macOS) either using the Configure button from the start screen or the IntelliJ IDEA menu.
- In the Settings/Preferences window, expand the Languages & Frameworks item and select XQuery.
- Make the choices for your system from the dropdown boxes, e.g.:
  - Implementation = BaseX
  - Implementation version = BaseX 9.1
  - Default XQuery version = XQuery 3.1
  - Dialect for XQuery 3.0 = BaseX
  - Dialect for XQuery 3.1 = BaseX
- Click Apply to store your XQuery settings and then OK to exit the dialog.

Querying Your Data

Create a New Project

- To create a new project choose the Create new project option from the start screen or select New → Project... from the File menu.
- In the New Project dialog, choose Empty Project from the left-hand column and click the Next button.
- Enter a name and location for your project and click on the Finish button.

Customize the XQuery Module

- Click the Add Configuration button below the IntelliJ IDEA menu bar.
Integrating IntelliJ IDEA

• In the Run/Debug Configurations dialog, expand the Templates list and choose the XQuery entry.
• Click on the three dots ... next to the Query Processor dropdown box.
• In the Manage Query Processors dialog, click on the + button.
• In the New Query Processor Instance dialog, set the following preferences:
  • Description = BaseX (optional; if you leave this field blank, [Implementation] [Version] will be used as description)
  • Implementation = BaseX (should be preset!)
  • JAR File = BaseX.jar (name and location of the JAR file may differ depending on your BaseX distribution and version)
  • Hostname = localhost
  • Database port = 1984
  • Username = admin
  • Password = ...
  • If you tick the check box Create a standalone instance, the fields Hostname, Database port, Username, and Password remain empty.

• Click OK to exit the New Query Processor Instance dialog.
• In the Manage Query Processors dialog, now choose the "BaseX [Version] (BaseX)" entry and click OK.
• The Query Processor dropdown box in the Run/Debug Configurations dialog should now also display "BaseX [Version] (BaseX)". If not, select it from the dropdown box.
• Click Apply and then OK to close the Run/Debug Configurations dialog.

Create a Query File

• In the project view, create a new XQuery file, either by right-clicking on the project name and choosing New File or by selecting New File from the File menu. Enter a file name and click OK.
• Type in your query, e.g. db:get("factbook"), and save your file.

Create a New Configuration
Integrating IntelliJ IDEA

• Click on the Add Configuration button once again.

• In the Run/Debug Configurations dialog, click the + button to create a new configuration based on a template.

• Choose the "XQuery" template you configured earlier.

• Enter a name, e.g. "BaseX", into the Name field.

• The query processor should be preset to "BaseX [Version] (BaseX)".

• In the Run the query from area, either enter the path to your query file into the Local file field to limit the run configuration to that query or choose the Active editor file option to make the configuration run the script that is currently opened in the IntelliJ editor panel.

• Click Apply and then OK to close the Run/Debug Configurations dialog.

• Now, the configuration should be set and the green Run button should be available below the IntelliJ IDEA menu bar.

Execute Your Query

• If the configuration does not run as a standalone instance, make sure that BaseX is up and running.

• Click the Run button to execute your query.

Conclusion

The plugin is very well maintained! It adds support for various XQuery Implementations to the IntelliJ IDEA (among them BaseX). It provides syntax highlighting for XQuery and XML, code completion and detects syntactical errors while you type offering a description for each error. Queries are executed using Run Configurations for which you can configure various query processors, e.g. BaseX.

BaseX's admin log can be accessed and displayed using the Query Log button on the bottom left corner of the IntelliJ IDEA project window.

The plugin contains some minor flaws regarding the use of functions declared in user-defined modules. Such functions are not included in the code completion list and marked as unknown in the code. However, query execution in the BaseX backend works fine nonetheless.

XQuery Support Plugin

This section focuses on Grzegorz Ligas' XQuery Support plugin.

Installation

After installing IntelliJ IDEA and BaseX, install the XQuery Support plugin by one of the following methods:

From the Start Screen
Integrating IntelliJ IDEA

• Start IntelliJ IDEA and select Configure → Plugins.

• In the Plugins window, select the tab Marketplace.

• Type "XQuery" into the Search plugins in marketplace field and press Enter.

• Click the Install button below the XQuery Support plugin or click on the XQuery Support link to get more information on the plugin before installing it.

• You will be prompted to restart IDEA to load the new plugin.

From the IntelliJ IDEA Menu

• Select Settings (Windows)/Preferences (macOS) from the IntelliJ IDEA menu.

• In the Settings/Preferences window, select Plugins.

• In the Plugins panel, select the tab Marketplace.

• Type "XQuery" into the Search plugins in marketplace field and press Enter.

• Click the Install button below the XQuery Support plugin or click on the XQuery Support link to get more information on the plugin before installing it.

• You will be prompted to restart IDEA to load the new plugin.

Setting Up

File Extensions and XQuery Flavor

• Start IntelliJ IDEA and navigate to Settings (Windows)/Preferences (macOS) either using the Configure button on the start screen or the IntelliJ IDEA menu.

• In the Settings/Preferences window, expand the Languages & Frameworks item, select XQuery and choose which default file extensions and which XQuery flavor you would like to use.

• Click Apply to store your XQuery settings.

Configuring The Processor

You can set up the plugin as a standalone processor or client.

Standalone

• In the Settings (Windows)/Preferences (macOS) window, expand the Languages & Frameworks item and select XQuery Data Sources.
Integrating IntelliJ IDEA

- Click on the + button in the middle column to add a new data source.

- Select *BaseX (native embedded)* from the dropdown box.

- In the right-hand column, check the *User defined XQJ Driver* check box.

- Use the + button below the check box to add the following jars from your BaseX distribution:
  
  - `basex/BaseX.jar`
  - `basex/lib/basex-apj-9.1.2.jar`
  - `basex/lib/basex-xqj-9.0.jar`
  - `basex/lib/xqj2-0.2.0.jar`

- Click *Apply* to store your settings.

**Client**

This assumes that you already have a BaseX database named *factbook*.

- In the *Settings* (Windows)/*Preferences* (macOS) window, expand the *Languages & Frameworks* item and select *XQuery Data Sources*.

- Click on the + button in the middle column to add a new data source.

- Select *BaseX* from the dropdown box.

- In the right-hand column, fill in the appropriate connection details, e.g. default values:
  
  - **Host** = localhost
  - **Port** = 1984
  - **Database name** = factbook
  - **Username** = admin
  - **Password** = ...

- Select *Apply*, then *OK* and your BaseX *factbook* database is ready to query.
Integrating IntelliJ IDEA

Querying Your Data

Create a New Project

• To create a new project choose the Create new project option from the start screen or select New → Project... from the File menu.
• In the New Project dialog, choose Empty Project from the left-hand column and click the Next button.
• Enter a name and location for your project and click on the Finish button.

Customize the XQuery Module

• Click the Add Configuration button below the IntelliJ IDEA menu bar.

• In the Run/Debug Configurations dialog, expand the Templates list and choose the XQuery Main Module entry.
• Click on the Configure button next to the Data Source field and either choose the previously configured standalone version (BaseX (native embedded) item) or the client version (BaseX item) from the list.
• Click Apply and then OK to close the Run/Debug Configurations dialog.

Create a Query File

• In the project view, create a new XQuery file by right-clicking on the project name and choosing New → XQuery File. Enter a file name, select Main Module from the Kind dropdown and click OK.
• Type in your query and save your file.

Create a New Configuration

• Click on the Add Configuration button once again.
• In the Run/Debug Configurations dialog, click the + button to create a new configuration based on a template.
• Choose the "XQuery Main Module" template you configured earlier.
• Enter a name, e.g. "BaseX", into the Name field.
• The data source should be preset either to "BaseX (native embedded)" or BaseX depending on your processor configuration.
• In the Main file field, enter the path to your query file.
• Click Apply and then OK to close the Run/Debug Configurations dialog.

• Now, the configuration should be set and the green Run button should be available below the IntelliJ IDEA menu bar.

Execute Your Query

• If the configuration does not run as a standalone instance, make sure that BaseX is up and running.

• Click the Run button to execute your query.

Conclusion

The plugin adds support for various XQuery Implementations to the IntelliJ IDEA (among them BaseX). It provides syntax highlighting for XQuery and XML and detects syntactical errors while you type offering a description for each error. Queries are executed using Run Configurations for which you can configure various query processors, e.g. BaseX. The plugin offers code completion for XQuery functions, integrated library modules, such as FunctX or the BaseX Module Library, and user-defined modules. IntelliJ’s Find Usages and Go To options seem to work fine for variables and functions, even across modules. Users can set XQuery-specific code style preferences.

This plugin also has a few minor drawbacks. If no path is specified, syntax highlighting marks user-defined modules as unknown, even if they reside in the designated BaseX module repository. However, the BaseX query processor, resolves them correctly during query execution. Error messages in the editor seem to be kept rather general and should me more specific. Parameter lists of code completion may be quite extensive and clog the screen. Leading tab space can be increased in user-defined steps, but neither decreased in single, nor user-defined steps.
Part VI. XQuery Portal
Chapter 20. 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. The processor is also a flexible general purpose processor, which can access and process local and remote sources and output results in various formats. BaseX is highly compliant with the official specifications. This section contains information on the query processor and its extensions:

- XQuery 3.0 and XQuery 3.1
  Features of the new XQuery Recommendations.

- XQuery Extensions and XQuery Optimizations
  Specifics of the BaseX XQuery processor.

- Module Library
  Additional functions included in the internal modules.

- Java Bindings
  Accessing and calling Java code from XQuery.

- Repository
  Install and manage XQuery and Java modules.

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

- Update
  Updating databases and local resources via XQuery Update.

- Indexes
  Available index structures and their utilization.

- Serialization
  Serialization parameters supported by BaseX.

- Errors
  Errors raised by XQuery expressions.
Chapter 21. XQuery 3.0

Enhanced FLWOR Expressions

Most clauses of FLWOR expressions can 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:

```xquery
for $country in db:get('factbook')//country
where $country/@population > 100000000
for $city in $country//city[population > 100000]
group by $name := $country/name[1]
count $id
return <country id='{ $id }' name='{ $name }'>{ $city/name }</country>
```

group by

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

XQuery:

```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:
XQuery 3.0

return element { $income } { $ppl }

Result:

<challenge>
  <person id="person0">
    <name>Kasidit Treweek</name>
  </person>
  ...<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
  return element persons {
    attribute country { $country },
    for $name in $person/@name
      return element name { data($name) }
  }

Result:

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

count

The 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 }">

allowing empty

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

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

window

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.
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:

  ```xquery```
  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:

  ```xquery```
  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.

  ```xquery```
  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.

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:**

```xquery```
(1 to 10) ! element node { . }
```xquery```

In contrast to path expressions, the results of the map operator will not be made duplicate-free and returned in document order.

Try/Catch

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

**Example:**

```xquery```
try {
```
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:**

```xquery
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:**

```xquery
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

**Expanded QNames**

A QName can be 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 \( \pi \)
• Q(java:java.io.FileOutputStream)new("output.txt") creates a new Java file output stream

Namespace Constructors

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

```xml
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 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.

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

Serialization

Serialization parameters can 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:

```xml
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
<?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 be specified in the prolog of an XQuery expression:

Example:

```xml
declare context item := document {
  <xml>
    <text>Hello</text>
    <text>World</text>
  </xml>
};
for $t in ./text()
return string-length($t)
```
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)
```

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:foreach
- fn:for-each-pair
- 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: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 8.4

- Added: %non-deterministic

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 22. Higher-Order Functions

Read this entry online in the BaseX Wiki.

This page present some higher-order functions of the XQuery specification. 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 will 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 behavior and abstract them into a library function.
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()*, $function as function(item()) as item()*) as item()*
```

**Summary**

Applies the specified $function to every item of $seq and returns all results as a single sequence.

**Examples**

- Square 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

- Apply 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

- Process each item of a sequence with the arrow operator:
  ```
  (*one", "two", "three") => fn:for-each(fn:upper-case(?))
  ```
  
  *Result:* ONE TWO THREE

**XQuery 1.0**

At the core, for-each is nothing else than a simple FLWOR expression:

```
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) {
  ```
Higher-Order Functions

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

Result: FocBar BAR
```

- Inefficient prime number generator:

```xquery
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:

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

XQuery 1.0

At the core, for-each is nothing else than a filter expression:

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

fn:for-each-pair

**Signatures**

```xquery
fn:for-each-pair($seq1 as item()*, $seq2 as item()*, $function as function(item(), item()) as item()*) as item()*
```

**Summary**

Applies the specified $function to the successive pairs of items of $seq1 and $seq2. Evaluation is stopped if one sequence yields no more items.

**Examples**

- Adding one to the numbers at odd positions:

```xquery
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:

```xquery
let $number-words := function($str) {
  fn:string-join(  
    fn:for-each-pair(    
      1 to 1000000000,    
      tokenize($str, ' +'),    
      concat(?-, ': ', ?)    
    ),    
    '
' )
}
```

118
Higher-Order Functions

```xquery
return $number-words('how are you?')
```

**Result:**

1: how
2: are
3: you?

- Checking if a sequence is sorted:

```xquery
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**

```xquery
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*:

```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 flavors. They differ in the direction in which they traverse the sequence:

**fn:fold-left**

**Signatures**

| fn:fold-left($seq as item()*, $seed as item()*, $function as function(item()*, item()) as item()*) as item()* |
Higher-Order Functions

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:

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

Examples

- Product of a sequence of integers:
  ```xml
  fn:fold-left(1 to 5, 1, 
    function($result, $curr) { $result * $curr }
  )
  Result:120
  ```

- Illustrating the evaluation order:
  ```xml
  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:
  ```xml
  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:

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

fn:fold-right

Signatures

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

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

Examples

- Product of a sequence of integers:
  ```xml
  fn:fold-right(1 to 5, 1, 
    function($curr, $result) { $result * $curr }
  )
  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()*,
    $function as function(item(), item()*) as item()*
) as item() {
    if(empty($seq)) then $seed
    else $function(
        fn:head($seq),
        local:fold-right(tail($seq), $seed, $function)
    )
}
```

Note that the order of the arguments of $fun are inverted compared to that in fn:fold-left(...).
Chapter 23. XQuery 3.1

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It provides a summary of the most important features of the XQuery 3.1 Recommendation.

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:

```xml
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:

```xquery
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:

```xml
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:

```xquery
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:

```xquery
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. 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:

```xquery
let $map := map { 'R': 'red', 'G': 'green', 'B': 'blue' }
return (  
    $map?*  (: 1. returns all values; same as: map:keys($map) ! $map(.) :) )
```
XQuery 3.1

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

**Arrow Operator**

The arrow operator `=>` provides a convenient alternative syntax for passing on functions to a value. The expression that precedes the operator will be supplied as first argument of the function that follows the arrow. 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 $f(v, j)$:

```
($v => f())
```

String Constructor

The string constructor has been inspired by here document literals of the Unix shell and script languages. It allows you to generate strings that contain various characters that would otherwise be interpreted as XQuery delimiters.

The string constructors syntax uses two backticks and a square bracket for opening and closing a string:

```
```
```
return ```«Count `{ $c }`, and I will be there.«```  

## Serialization

Two Serialization methods have been added to the Serialization spec:

### Adaptive Serialization

The adaptive serialization provides an intuitive textual representation for all XDM types, including maps and arrays, functions, attributes, and namespaces. All items will be separated by the value of the `item-separator` parameter, which by default is a newline character. It is utilized by the functions `prof:dump` and `fn:trace`.

**Example:**

```xml
declare option output:method 'adaptive';
<element id='id0'/>/@id,
xs:token("abc"),
map { 'key': 'value' },
true#0
```

**Result:**

```xml
id="id0"
xs:token("abc"),
map {
   "key": "value"
} fn: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 before it was added to the official specification. It 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" }`:

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

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

## Functions

The following functions have been added in the XQuery 3.1 Functions and Operators Specification:

### Map Functions

`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**


**JSON Functions**

With XQuery 3.1, native support for JSON objects was added. 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.

<table>
<thead>
<tr>
<th>parse-json('{ &quot;name&quot;: &quot;john&quot; }')   (: yields { &quot;name&quot;: &quot;json&quot; } :)</th>
<th>parse-json('[ 1, 2, 4, 8, 16]')    (: yields [ 1, 2, 4, 8, 16 ] :)</th>
</tr>
</thead>
</table>

**fn:json-doc**

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).

<table>
<thead>
<tr>
<th>json-doc(&quot;<a href="http://ip.jsontest.com/%22)('ip">http://ip.jsontest.com/&quot;)('ip</a>')  (: returns your IP address :)</th>
</tr>
</thead>
</table>

**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.

<table>
<thead>
<tr>
<th>json-to-xml('{ &quot;message&quot;: &quot;world&quot; }')</th>
</tr>
</thead>
</table>

| (: result: |
| <map xmlns="http://www.w3.org/2005/xpath-functions"> |
|  <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.
XQuery 3.1

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

**fn:sort**

Signatures

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

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sort(reverse(1 to 3))</code></td>
<td>yields 1, 2, 3</td>
</tr>
<tr>
<td><code>reverse(sort(1 to 3))</code></td>
<td>returns the sorted order in descending order</td>
</tr>
<tr>
<td><code>sort((3,-2,1), (), abs#1)</code></td>
<td>yields 1, -2, 3</td>
</tr>
<tr>
<td><code>sort((1,2,3), (), function($x) { -$x })</code></td>
<td>yields 3, 2, 1</td>
</tr>
<tr>
<td><code>sort((1,'a'))</code></td>
<td>yields an error, as strings and integers cannot be compared</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contains-token(('a', 'b c', 'd'), 'c')</code></td>
<td>yields true</td>
</tr>
<tr>
<td><code>&lt;xml class='one two'/&gt;/contains-token(@class, 'one')</code></td>
<td>yields true</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fn:parse-ietf-date('Wed, 01 Jun 2001 23:45:54 +02:00')&quot;</code></td>
<td>yields 2001-06-01T23:45:54+02:00</td>
</tr>
</tbody>
</table>

**fn:apply**

Signatures

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

The supplied $function is invoked with the specified $arguments. The arity of the function must be the same as the size of the array.
Example:

```xquery
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:**

```xquery
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/>" :)  

### fn:string-join

The type of the first argument is now `xs:anyAtomicType*`, and all items will be implicitly cast to strings:

- `fn:string-join(1 to 3)`  (: yields the string "123" :)

---

128
**fn:default-language**

Returns the default language used for formatting numbers and dates. BaseX always returns en.

**Appendix**

The three functions fn:transform, fn:load-xquery-module and fn:collation-key may be added in a future version of BaseX as their implementation might require the use of additional external libraries.

**Binary Data**

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

\[
\begin{align*}
\text{xs:hexBinary('') & < xs:hexBinary('bb'),} \\
\text{xs:hexBinary('aa') & < xs:hexBinary('bb'),} \\
\text{max((xs:hexBinary('aa'), xs:hexBinary('bb'))) & = xs:hexBinary('bb')} 
\end{align*}
\]

**Collations**

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

```xml
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 become available in BaseX:

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

**Enclosed Expressions**

Enclosed expression is the syntactical term for the expressions that are specified inside a function body, try/catch clauses, node constructors and some other expressions. In the following example expressions, its the empty sequence:

```xml
declare function local:x() { () };
try { () } catch * { () },
element x { () },
text { () }
```

With XQuery 3.1, the expression can be omitted. The following query is equivalent to the upper one:

```xml
declare function local:x() { };
try { } catch * { },
element x { } 
text { }
```

**Changelog**

Version 8.6

- Updated: Collation argument was inserted between first and second argument.

Version 8.4

- Added: String Constructors, fn:default-language, Enclosed Expressions
- Updated: Adaptive Serialization, fn:string-join
Version 8.2


Version 8.1

• Updated: arrays are now based on a Finger Tree implementation.

Introduced with Version 8.0.
Chapter 24. XQuery Extensions

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It lists extensions and optimizations that are specific to the BaseX XQuery processor.

Expressions

Some of the extensions that have been added to BaseX may also be made available in other XQuery processors in the near future.

Ternary If

The ternary if operator provides a short syntax for conditions. It is also called conditional operator or ternary operator. In most languages, the syntax is `a ? b : c`. As `?` and `:` have already been taken in XQuery, the syntax of Perl 6 is used:

```
@test ?? 'ok' !! 'fails'
```

The expression returns `ok` if the effective boolean value of `$test` is true, and it returns `fails` otherwise.

Elvis Operator

The Elvis operator is also available in other languages. It is sometimes called null-coalescing operator. In XQuery, the value of the first operand will be returned if it is a non-empty sequence. Otherwise, the value of the second operand will be returned.

```xquery
let $number := 123
return (
    (: if/then/else :) if (exists($number)) then $number else 0,
    (: elvis operator :) $number ?: 0
)
```

The behavior of the operator is equivalent to the `util:or` function.

If Without Else

In XQuery 3.1, both branches of the `if` expression need to be specified. In many cases, only one branch is required, so the `else` branch was made optional in BaseX. If the second branch is omitted, an empty sequence will be returned if the effective boolean value of the test expression is false. Some examples:

```xquery
if (doc-available($doc)) then doc($doc),
if (file:exists($file)) then file:delete($file),
if (permissions:valid($user)) then <html>Welcome!</html>
```

If conditions are nested, a trailing else branch will be associated with the innermost `if`:

```
if ($a) then if($b) then '$a and $b is true' else 'only $a is true'
```

In general, if you have multiple or nested if expressions, additional parentheses can improve the readability of your code:

```
if ($a) then ( if($b) then '$a and $b is true' else 'only $a is true' )
```

The behavior of the if expression is equivalent to the `util:if` function.
Functions

Regular Expressions

In analogy with Saxon, you can specify the flag `j` to revert to Java’s default regex parser. For example, this allows you to use the word boundary option `\b`, which has not been included in the XQuery grammar for regular expressions:

Example:

```
(: yields "!Hi! !there!" :)
replace('Hi there', '\b', '!', 'j')
```

Serialization

- `basex` is used as the default serialization method: nodes are serialized as XML, atomic values are serialized as string, and items of binary type are output in their native byte representation. Function items (including maps and arrays) are output just like with the `adaptive` method.

- With `csv`, you can output XML nodes as CSV data (see the CSV Module for more details).

- With `json`, items are output as JSON as described in the official specification. If the root node is of type `element(json)`, items are serialized as described for the `direct` format in the JSON Module.

For more information and some additional BaseX-specific parameters, see the article on Serialization.

Option Declarations

Database Options

Local database options can be set in the prolog of an XQuery main module. 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:catalog 'etc/w3-catalog.xml';
doc('doc.xml')
```

XQuery Locks

If locks are declared in the query prolog of a module via the `basex:lock` option, access to functions of this module locks will be controlled by the central transaction management. See Transaction Management for further details.

Pragmas

BaseX Pragmas

Many optimizations in BaseX will only be performed if an expression is deterministic (i.e., if it always yields the same output and does not have side effects). By flagging an expression as non-deterministic, optimizations and query rewritings can be suppressed:

```
sum( { # basex:non-deterministic # } { 1 to 10000000 })
```

This pragma can be helpful when debugging your code.

In analogy with option declarations and function annotations, XQuery locks can also set via pragmas. See Transaction Management for details and examples.
### Database Pragmas

Local database options can also be assigned via pragmas:

- **Index access rewritings** can be enforced. This is helpful if the name of a database is not static (see Enforce Rewritings for more details):

  ```xml
  (% db:enforceindex #) {
  for $db in ('persons1', 'persons2', 'persons3')
  return db:get($db)//name[text() = 'John']
  }
  ```

- **Node copying in node constructors** can be disabled (see COPYNODE for more details). The following query will consume much less memory than without pragma as the database nodes will not be fully duplicated, but only attached to the xml parent element:

  ```xml
  file:write('wrapped-db-nodes.xml',
  (% db:copynode false #) {
  <xml>{ db:get('huge') } </xml>
  }
  )
  ```

- An XML catalog can be specified for URI rewritings. See the Catalog Resolver section for an example.

### Annotations

#### Function Inlining

%basex:inline([limit]) controls if functions will be inlined.

If XQuery functions are *inlined*, the function call will be replaced by a FLWOR expression, in which the function variables are bound to let clauses, and in which the function body is returned. This optimization triggers further query rewritings that will speed up your query. An example:

**Query:**

```xml
declare function local:square($a) { $a * $a };
for $i in 1 to 3
return local:square($i)
```

**Query after function inlining:**

```xml
for $i in 1 to 3
return
  let $a := $i
  return $a * $a
```

**Query after further optimizations:**

```xml
for $i in 1 to 3
return $i * $i
```

By default, XQuery functions will be *inlined* if the query body is not too large and does not exceed a fixed number of expressions, which can be adjusted via the INLINELIMIT option.

The annotation can be used to overwrite this global limit: Function inlining can be enforced if no argument is specified. Inlining will be disabled if 0 is specified.
XQuery Extensions

Example:

(: disable function inlining; the full stack trace will be shown... :)  
declare %basex:inline(0) function local:e() { error() };  
local:e()

Result:

Stopped at query.xq, 1/53:  
[FOER0000] Halted on error().

Lazy Evaluation

%basex:lazy enforces lazy evaluation of a global variable. An example:

Example:

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

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

XQuery Locks

In analogy with option declarations and pragmas, locks can also set via annotations. See Transaction Management for details and examples.

Non-Determinism

In XQuery, deterministic functions are “guaranteed to produce identical results from repeated calls within a single execution scope if the explicit and implicit arguments are identical”. In BaseX, many extension functions are non-deterministic or side-effecting. If an expression is internally flagged as non-deterministic, various optimizations that might change their execution order will not be applied.

( : QUERY A... : )  
let $n := 456  
for $i in 1 to 2  
return $n

( : ...will be optimized to : )  
for $i in 1 to 2  
return 456

( : QUERY B will not be rewritten : )  
let $n := random:integer()  
for $i in 1 to 2  
return $n

In some cases, functions may contain non-deterministic code, but the query compiler may not be able to detect this statically. See the following example:

for $read in (file:read-text#1, file:read-binary#1)  
let $ignore := non-deterministic $read('input.file')  
return ()

Two non-deterministic functions will be bound to $read, and the result of the function call will be bound to $ignore. As the variable is not referenced in the subsequent code, the let clause would usually be discarded by the compiler. In the given query, however, execution will be enforced because of the BaseX-specific non-deterministic keyword.
Namespaces

In XQuery, some namespaces are statically bound to prefixes. The following query requires no additional
namespaces declarations in the query prolog:

```xml
<xml:abc xmlns:prefix='uri' local:fn='x'/>,
fn:exists(1)
```

In BaseX, various other namespaces are predefined. Apart from the namespaces that are listed on the Module
Library page, the following namespaces are statically bound:

<table>
<thead>
<tr>
<th>Description</th>
<th>Prefix</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseX Annotations, Pragmas, …</td>
<td>basex</td>
<td><a href="http://basex.org">http://basex.org</a></td>
</tr>
<tr>
<td>RESTXQ: Input Options</td>
<td>input</td>
<td><a href="http://basex.org/modules/input">http://basex.org/modules/input</a></td>
</tr>
<tr>
<td>EXPath Packages</td>
<td>pkg</td>
<td><a href="http://expat.org/ns/pkg">http://expat.org/ns/pkg</a></td>
</tr>
<tr>
<td>XQuery Errors</td>
<td>err</td>
<td><a href="http://www.w3.org/2005/xqt-errors">http://www.w3.org/2005/xqt-errors</a></td>
</tr>
<tr>
<td>Serialization</td>
<td>output</td>
<td><a href="http://www.w3.org/2010/xslt-xquery-serialization">http://www.w3.org/2010/xslt-xquery-serialization</a></td>
</tr>
</tbody>
</table>

Suffixes

In BaseX, files with the suffixes .xq, .xqm, .xqy, .xql, .xqu and .xquery are treated as XQuery files. In
XQuery, there are main and library modules:

- Main modules have an expression as query body. Here is a minimum example:

  ```
  'Hello World!
  ```

- Library modules start with a module namespace declaration and have no query body:

  ```
  module namespace hello = 'http://basex.org/examples/hello';
  declare function hello:world() {
      'Hello World!
  };
  ```

We recommend .xq as suffix for for main modules, and .xqm for library modules. However, the actual module
type will dynamically be detected when a file is opened and parsed.

Miscellaneous

Various other extensions are described in the articles on XQuery Full Text and XQuery Update.

Changelog

Version 9.1

- Added: New Expressions: Ternary if, elvis Operator, if without else
- Added: XQuery Locks via pragmas and function annotations.
- Added: Regular Expressions, j flag for using Java’s default regex parser.
Chapter 25. XQuery Optimizations

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It presents some of the optimizations that speed up the execution and reduce memory consumption of queries.

Introduction

Query execution encompasses multiple steps:

1. Parsing: The query input string is transformed to executable code. The result is a tree representation, called the abstract syntax tree (AST).

2. Compilation: The syntax tree is decorated with additional information (type information, expression properties). Expressions (nodes) in the tree are relocated, simplified, or pre-evaluated. Logical optimizations are performed that do not rely on external information.

3. Optimization: The dynamic context is incorporated: Referenced databases are opened and analyzed; queries are rewritten to use available indexes; accumulative and statistical operations (counts, summations, min/max, distinct values) are pre-evaluated; XPath expressions are simplified, based on the existence of steps.

4. Evaluation: The resulting code is executed.

5. Printing: The query result is serialized and presented in a format that is either human-readable, or can be further processed by an API.

Some rewritings are described in this article.

If you run a query on command-line, you can use \(-V\) to output detailed query information. In the GUI, you can enable the Info View panel.

Compilation

Pre-Evaluation

Parts of the query that are static and would be executed multiple times can already be evaluated at compile time:

```
for $i in 1 to 10
return 2 * 3

(: rewritten to :)
for $i in 1 to 10
return 6
```

Variable Inlining

The value of a variable can be inlined: The variables references are replaced by the expression that is bound to the variable. The resulting expression can often be simplified, and further optimizations can be triggered:

```
declare variable $INFO := true();

let $nodes := //nodes
where $INFO
return 'Results: ' || count($nodes)

(: rewritten to :)
let $nodes := //nodes
where true()
return 'Results: ' || count($nodes)
```
As the example shows, variable declarations might be located in the query prolog and in FLWOR expressions. They may also occur (and be inlined) in `try/catch`, `switch` or `typeswitch` expressions.

**Function Inlining**

Functions can be inlined as well. The parameters are rewritten to `let` clauses and the function is body is bound to the `return` clause.

```xquery
declare function local:inc($i) { $i + 1 }; for $n in 1 to 5
return local:inc($n)
```

(: rewritten to :) for $n in 1 to 5 return ($n + 1)

Subsequent rewritings might result in query plans that differ a lot from the original query. As this might complicate debugging, you can disable function inlining during development by setting `INLINELIMIT` to 0.

**Loop Unrolling**

Loops with few iterations are **unrolled** by the XQuery compiler to enable further optimizations:

```xquery
(1 to 2) ! (. * 2)
```

(: rewritten to :) 1 ! (. * 2), 2 ! (. * 2)

(: further rewritten to :) 1 * 2, 2 * 2

Folds are unrolled, too:

```xquery
let $f := function($a, $b) { $a * $b } return fold-left(2 to 5, 1, $f)
```

(: rewritten to :) let $f := function($a, $b) { $a * $b } return $f($f($f($f(1, 2), 3), 4), 5)

The standard unroll limit is 5. It can be adjusted with the `UNROLLLIMIT` option, e.g. via a pragma:

```xquery
(# db:unrolllimit 10 #) {
  for $i in 1 to 10
  return db:get('db' || $i)//*[text() = 'abc']
}
```

(: rewritten to :)
The last example indicates that index rewritings might be triggered by unrolling loops with paths on database nodes. The following expressions can be unrolled:

- Simple map expressions
- Simple FLWOR expressions
- Filter expressions
- `fn:fold-left`, `fn:fold-right`, `fn:fold-left1`

Care should be taken if a higher value is selected, as memory consumption and compile time will increase.

### Paths

Due to the compact syntax of XPath, it can make a big difference if a slash is added or omitted in a path expression. A classical example is the double slash `//`, which is a shortcut for `descendant-or-node()`/. If the query is evaluated without optimizations, all nodes of a document are gathered, and for each of them, the next step is evaluated. This leads to a potentially huge number of duplicate node tree traversals, most of which are redundant, as all duplicate nodes will be removed at the end anyway.

In most cases, paths with a double slash can be rewritten to descendant steps…

```xml
(doc('addressbook.xml')//city,
doc('addressbook.xml')/descendant-or-self::node()/child::city

...unless the last step does not contain a positional predicate:

doc('addressbook.xml')//city[1]
```

As the positional test refers to the city child step, a rewritten query would yield different steps.

Paths may contain predicates that will be evaluated again by a later axis step. Such predicates are either shifted down or discarded:

```xml
(a[b]/b[c[d] / c

...rewritten to:

a/b/c[d]
```

Names of nodes can be specified via name tests or predicates. If names are e.g. supplied via external variables, the predicates can often be dissolved:

```xml
declare variable $name external := 'city';
db:get('addressbook')/descendant::*[name() = $name]
```

### FLWOR Rewritings

FLWOR expressions are central to XQuery and the most complex constructs the language offers. Numerous optimizations have been realized to improve the execution time:
XQuery Optimizations

- Nested FLWOR expressions are flattened.
- for clauses with single items are rewritten to let clauses.
- let clauses that are iterated multiple times are lifted up.
- Expressions of let clauses are inlined.
- Unused variables are removed.
- where clauses are rewritten to predicates.
- if expressions in the return clause are rewritten to where clauses.
- The last for clause is merged into the return clause and rewritten to a simple map expression.

Various of these rewriting are demonstrated in the following example:

```xquery
for $a in 1 to 10
  for $b in 2
    where $a > 3
    let $c := $a + $b
  return $c
```

(: for is rewritten to let :)  
```
for $a in 1 to 10
  let $b := 2
  where $a > 3
  let $c := $a + $b
  return $c
```

(: let is lifted up :)  
```
let $b := 2
for $a in 1 to 10
  where $a > 3
  let $c := $a + $b
  return $c
```

(: the where expression is rewritten to a predicate :)  
```
let $b := 2
for $a in (1 to 10)[. > 3]
  let $c := $a + $b
  return $c
```

(: $b is inlined :)  
```
for $a in (1 to 10)[. > 3]
  let $c := $a + 2
  return $c
```

(: $c is inlined :)  
```
for $a in (1 to 10)[. > 3]
  return $a + 2
```

(: the remaining clauses are merged and rewritten to a simple map :)  
```
(1 to 10)[. > 3] ! ( . + 2)
```

Static Typing

If the type of a value is known at compile time, type checks can be removed. In the example below, the static information that $i will always reference items of type xs:integer can be utilized to simplify the expression:

```xquery
for $i in 1 to 5
  return typeswitch($i)
    case xs:numeric return 'number'
```
XQuery Optimizations

default return 'string'

(; rewritten to ;)
for $i in 1 to 5
return 'number'

Pure Logic

If expressions can often be simplified:

for $a in ('a', '')
return $a[boolean(if(.) then true() else false())]

(; rewritten to ;)
for $a in ('a', '')
return $a[boolean(.)]

(; rewritten to ;)
('a', '')[.]

Boolean algebra (and set theory) comes with a set of laws that can all be applied to XQuery expressions.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rewritten expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + 0, $a * 1</td>
<td>$a</td>
<td>Identity</td>
</tr>
<tr>
<td>$a * 0</td>
<td>0</td>
<td>Annihilator</td>
</tr>
<tr>
<td>$a and $a</td>
<td>$a</td>
<td>Idempotence</td>
</tr>
<tr>
<td>$a and ($a or $b)</td>
<td>$a</td>
<td>Absorption</td>
</tr>
<tr>
<td>($a and $b) or ($a and $c)</td>
<td>$a and ($b or $c)</td>
<td>Distributivity</td>
</tr>
<tr>
<td>$a or not($a)</td>
<td>true()</td>
<td>Tertium non datur</td>
</tr>
<tr>
<td>not($a) and not($b)</td>
<td>not($a or $b)</td>
<td>De Morgan</td>
</tr>
</tbody>
</table>

It is not sufficient to apply the rules to arbitrary input. Examples:

- If the operands are no boolean values, a conversion is enforced: $string and $string is rewritten to boolean($string).
- xs:double('NaN') * 0 yields NaN instead of 0
- true#0 and true#0 must raise an error; it cannot be simplified to true#0

Optimization

Some physical optimizations are also presented in the article on index structures.

Database Statistics

In each database, metadata is stored that can be utilized by the query optimizer to speed up or even skip query evaluation:

Count element nodes

The number of elements that are found for a specific path need not be evaluated sequentially. Instead, the count can directly be retrieved from the database statistics:

count(/mondial/country)
Return distinct values

The distinct values for specific names and paths can also be fetched from the database metadata, provided that the number does not exceed the maximum number of distinct values (see MAXCATS for more information):

```
distinct-values(//religions)
```

(: rewritten to :)

('Muslim', 'Roman Catholic', 'Albanian Orthodox', ...)

**Index Rewritings**

A major feature of BaseX is the ability to rewrite all kinds of query patterns for index access.

The following queries are all equivalent. They will be rewritten to exactly the same query that will eventually access the text index of a `factbook.xml` database instance (the file included in our full distributions):

```
declare context item := db:get('factbook');
declare variable $DB := 'factbook';

//name[.= 'Shenzhen'],
//name[data() = 'Shenzhen'],
//name[./text() = 'Shenzhen'],
//name[text()[. = 'Shenzhen']],
//name[string() = 'Shenzhen'],
//name[./data(text()/string()) = 'Shenzhen'],
//name[text() ! data() ! string() = 'Shenzhen'],
//name[. eq 'Shenzhen'],
//name[not(. ne 'Shenzhen')],
//name[not(. != 'Shenzhen')],
//*[local-name() = 'name'][data() = 'Shenzhen'],
db:get('factbook')//name[. = 'Shenzhen'],
db:get($DB)//name[. = 'Shenzhen'],
for $name in //name[text() = 'Shenzhen']
return $name,
for $name in //name
return $name[text() = 'Shenzhen'],
for $name in //name
return if($name/text() = 'Shenzhen') then $name else (),
for $name in //name
where $name/text() = 'Shenzhen'
return $name,
for $name in //name
where $name/text()[. = 'Shenzhen']
return $name,
for $node in */
where data($node) = 'Shenzhen'
where name($node) = 'name'
return $node,
```

141
XQuery Optimizations

Multiple element names and query strings can be supplied in a path:

```xquery
//*[[(ethnicgroups, religions)/text() = ('Jewish', 'Muslim')]]
```

If multiple candidates for index access are found, the database statistics (if available) are consulted to choose the cheapest candidate:

```xquery
/mondial/country
[religions = 'Muslim'] (: yields 77 results :) 
[ethnicgroups = 'Greeks'] (: yields 2 results :) 
```

If index access is possible within more complex FLWOR expressions, only the paths will be rewritten:

```xquery
for $country in //country
where $country/ethnicgroups = 'German'
order by $country/name[1]
return element { replace($country/@name, '', '') } {}
```

The XMark XML Benchmark comes with sample auction data and a bunch of queries, some of which are suitable for index rewritings:

XMark Query 1

```xquery
let $auction := doc('xmark')
return for $b in $auction/site/people/person[@id = 'person0']
return $b/name/text()
```

XMark Query 8

```xquery
let $auction := doc('xmark')
return
for $p in $auction/site/people/person
let $a :=
for $t in $auction/site/closed_auctions/closed_auction
where $t/buyer/@person = $p/@id
return $t
return <item person="( $p/name/text() )">{ count($a) }</item>,
```

---

142
If the accessed database is not known at compile time, or if you want to give a predicate preference to another one, you can enforce index rewritings.

**Evaluation**

**Comparisons**

In many cases, the amount of data to be processed is only known after the query has been compiled. Moreover, the data that is looped through expressions may change. In those cases, the best optimizations needs to be chosen at runtime.

If sequences of items are compared against each other, a dynamic hash index will be generated, and the total number of comparisons can be significantly reduced. In the following example, `count($input1) * count($input2)` comparisons would need to be made without the intermediate index structure:

```xml
let $input1 := file:read-text-lines('huge1.txt')
let $input2 := file:read-text-lines('huge2.txt')
return $input1[not(. = $input2)]
```

**Changelog**

Version 9.6

- Added: UNROLLLIMIT

Introduced with Version 9.4.
Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal.

In addition to the standard XQuery Functions, BaseX comes with hundreds of additional functions, which are packaged in various modules.

The namespaces of the built-in modules are statically bound to their prefix. This means that they need not (but may) be declared in the query prolog.

## Conventions

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

144
<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Module URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Client</td>
<td>Sending HTTP requests, HTTP based on the EXPath HTTP module.</td>
<td><a href="http://expath.org/ns/http-client">http://expath.org/ns/http-client</a></td>
</tr>
<tr>
<td>Index</td>
<td>Functions for requesting index details on database indexes.</td>
<td><a href="http://basex.org/modules/index">http://basex.org/modules/index</a></td>
</tr>
<tr>
<td>Inspection</td>
<td>Functions for extracting inspect internal module information.</td>
<td><a href="http://basex.org/modules/inspect">http://basex.org/modules/inspect</a></td>
</tr>
<tr>
<td>JSON</td>
<td>Parsing and serializing JSON documents.</td>
<td><a href="http://basex.org/modules/json">http://basex.org/modules/json</a></td>
</tr>
<tr>
<td>Lazy</td>
<td>Functions for handling lazy items.</td>
<td><a href="http://basex.org/modules/lazy">http://basex.org/modules/lazy</a></td>
</tr>
<tr>
<td>Map</td>
<td>Functions for handling maps (key/value pairs).</td>
<td><a href="http://www.w3.org/2005/xpath-functions/map">http://www.w3.org/2005/xpath-functions/map</a></td>
</tr>
<tr>
<td>Process</td>
<td>Executing system commands from XQuery.</td>
<td><a href="http://basex.org/modules/proc">http://basex.org/modules/proc</a></td>
</tr>
<tr>
<td>Profiling</td>
<td>Functions for profiling code snippets.</td>
<td><a href="http://basex.org/modules/prof">http://basex.org/modules/prof</a></td>
</tr>
<tr>
<td>Random</td>
<td>Functions for creating random numbers.</td>
<td><a href="http://basex.org/modules/random">http://basex.org/modules/random</a></td>
</tr>
<tr>
<td>SQL</td>
<td>JDBC bridge to access relational databases.</td>
<td><a href="http://basex.org/modules/sql">http://basex.org/modules/sql</a></td>
</tr>
<tr>
<td>Store</td>
<td>Organize values in a main-memory key-value store.</td>
<td><a href="http://basex.org/modules/store">http://basex.org/modules/store</a></td>
</tr>
<tr>
<td>String</td>
<td>Functions for performing string computations.</td>
<td><a href="http://basex.org/modules/string">http://basex.org/modules/string</a></td>
</tr>
<tr>
<td>Unit</td>
<td>Unit testing framework.</td>
<td><a href="http://basex.org/modules/unit">http://basex.org/modules/unit</a></td>
</tr>
<tr>
<td>User</td>
<td>Creating and administering database users.</td>
<td><a href="http://basex.org/modules/user">http://basex.org/modules/user</a></td>
</tr>
</tbody>
</table>
## Module Library

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utility</td>
<td>Various utility and helper functions.</td>
<td>[<a href="http://basex.org/modules/util">http://basex.org/modules/util</a>]</td>
</tr>
<tr>
<td>Validation</td>
<td>Validating documents: validate DTDs, XML Schema, RelaxNG.</td>
<td>[<a href="http://basex.org/modules/validate">http://basex.org/modules/validate</a>]</td>
</tr>
<tr>
<td>Web</td>
<td>Convenience functions for building web applications.</td>
<td>[<a href="http://basex.org/modules/web">http://basex.org/modules/web</a>]</td>
</tr>
<tr>
<td>XQuery</td>
<td>Evaluating new XQuery expressions at runtime.</td>
<td>[<a href="http://basex.org/modules/xquery">http://basex.org/modules/xquery</a>]</td>
</tr>
<tr>
<td>XSLT</td>
<td>Stylesheet transformations, xslt based on Java’s and Saxon’s XSLT processor.</td>
<td>[<a href="http://basex.org/modules/xslt">http://basex.org/modules/xslt</a>]</td>
</tr>
<tr>
<td>Request</td>
<td>Server-side functions for handling HTTP Request data.</td>
<td>[<a href="http://exquery.org/ns/request">http://exquery.org/ns/request</a>]</td>
</tr>
<tr>
<td>RESTXQ</td>
<td>Helper functions for the RESTXQ API.</td>
<td>[<a href="http://exquery.org/ns/restxq">http://exquery.org/ns/restxq</a>]</td>
</tr>
<tr>
<td>Session</td>
<td>Functions for handling server-side HTTP Sessions.</td>
<td>[<a href="http://basex.org/modules/session">http://basex.org/modules/session</a>]</td>
</tr>
<tr>
<td>Sessions</td>
<td>Functions for managing all server-side HTTP Sessions.</td>
<td>[<a href="http://basex.org/modules/sessions">http://basex.org/modules/sessions</a>]</td>
</tr>
<tr>
<td>WebSocket</td>
<td>Functions for handling WebSocket connections.</td>
<td>[<a href="http://basex.org/modules/ws">http://basex.org/modules/ws</a>]</td>
</tr>
</tbody>
</table>

The following modules are available if the `basex-api` library is included in the classpath. This is the case if you start BaseX with one of the startup scripts or links provided by our complete distributions (zip, exe, war).

## Changelog

### Version 10

- Removed: ZIP Module; Geo Module; Output Module (incorporated in String Module)
Chapter 27. Java Bindings

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It demonstrates different ways to invoke Java code from XQuery, and it presents extensions to access the current query context from Java.

The Java Binding feature is an extensibility mechanism which enables developers to directly access Java variables and execute code from XQuery. Addressed Java code must either be contained in the Java classpath, or it must be located in the Repository.

Please bear in mind that the execution of Java code may cause side effects that conflict with the functional nature of XQuery, or may introduce new security risks to your project.

Some more notes:

• With the middle dot notation, three adjacent dots can be used to specify array types.
• The path to the standard package java.lang. can be omitted.
• Java objects are wrapped into function items.
• Results of constructor calls are always returned as function item.
• With \texttt{WRAPJAVA}, it can be controlled how Java values are converted to XQuery.

Identification

Classes

A Java class is identified by a namespace URI. The original URI is rewritten as follows:

1. The URI Rewriting steps are applied to the URI.
2. Slashes in the resulting URI are replaced with dots.
3. The last path segment of the URI is capitalized and rewritten to CamelCase.

The normalization steps are skipped if the URI is prefixed with \texttt{java:}. The path to the standard package \texttt{java.lang.} can be omitted:

• \texttt{http://basex.org/modules/meta-data} \rightarrow \texttt{org.basex.modules.MetaData}
• \texttt{java:java.lang.String} \rightarrow \texttt{java.lang.String}
• \texttt{StringBuilder} \rightarrow \texttt{java.lang.StringBuilder}

Functions and Variables

Java constructors, functions and variables can be referenced and evaluated by the existing XQuery function syntax:

• The namespace of the function name identifies the Java class.
• The local part of the name, which is rewritten to camel case, identifies a variable or function of that class.
• The middle dot character \texttt{·} (\#x00B7;: a valid character in XQuery names, but not in Java) can be used to append exact Java parameter types to the function name. Class types must be referenced by their full path. Three adjacent dots can be used to address an array argument.

<table>
<thead>
<tr>
<th>Addressed code</th>
<th>XQuery</th>
<th>Java</th>
</tr>
</thead>
</table>

147
As XQuery and Java have different type systems, XQuery arguments must be converted to equivalent Java values, and the result of a Java function is converted back to an XQuery value (see Data Types).

If the Java function you want to address is not detected, you may need to cast your values to the target type. For example, if a Java function expects a primitive \texttt{int} value, you will need to convert your XQuery integers to \texttt{xs:int}.

### Namespace Declarations

In the following example, the Java \texttt{Math} class is referenced. When executed, the query returns the cosine of an angle by calling the static method \texttt{cos()}, and the value of $\pi$ by addressing the static variable via \texttt{PI()}:

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

With the Expanded QName notation of XQuery 3.0, the namespace can directly be embedded in the function call:

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

The constructor of a class can be invoked by calling the virtual function \texttt{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 \texttt{output.txt}. First, a new \texttt{FileWriter} instance is created, and its \texttt{write()} function is called in the next step:

```xquery
declare namespace fw = 'java: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))
```

If the result of a Java call contains invalid XML characters, it will be rejected. The validity check can be disabled by setting \texttt{CHECKSTRINGS} to false. In the example below, a file with a single \texttt{00} byte is written, and this file will then be accessed by via Java functions:

```xquery
declare namespace br = 'java:java.io.BufferedReader';
declare namespace fr = 'java:java.io.FileReader';
declare option db:checkstrings 'false';
(: write file :)file:write-binary('00.bin', xs:hexBinary('00')),
(: read file :)let $br := br:new(fr:new('00.bin'))
return (br:readLine($br), br:close($br))
```

The option can also be specified via a pragma:

```xquery
(# db:checkstrings #) { br:new(fr:new('00.bin')) ! (br:readLine(.), br:close(.))}
```
Module Imports

A Java class can be instantiated by importing them as a module: A new instance of the addressed class will be constructed, which can then be referenced in the query body.

In the (side-effecting) example below, a HashSet instance is created, values are added, and the size of the set is returned. As `set:add()` returns boolean values, `prof:void` is used to swallow the values:

```java
import module namespace set = "java:java.util.HashSet";
prof:void(
    for $s in ("one", "two", "one")
        return set:add($s)
),
set:size()
```

The execution of imported classes is more efficient than the execution of instances that have been created via `new()`. In turn, no arguments can be supplied in the import statement, and the construction will only be successful if the class can be instantiated without arguments.

Integration

Java classes can be coupled more closely to BaseX. 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.

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 the XQuery expression has been fully evaluated.

Annotations

The internal properties of functions can be assigned via annotations:

- Java functions can only be executed by users with Admin permissions. You can annotate a function with `@Requires(<Permission>)` to also make it accessible to users with fewer 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`

In the following code, information from the static query context is returned by the first function, and a query exception is raised by the second function:

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

element user {
    context:user()
},
try {
    element to-int { context:to-int('abc') }
} catch basex:error {
    element error { $err:description }
}
```

The imported Java class is shown below:

```java
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 string
     */
    @Requires(Permission.NONE)
    @Deterministic
    @ContextDependent
    public String user() {
        return queryContext.context.user.name;
    }

    /**
     * Converts the specified string to an integer.
     * @param value string to be converted
     * @return resulting 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("Integer conversion failed: " + value);
        }
    }

    @Override
    public void close() {
        // defined in QueryResource interface, will be called after query evaluation
    }
}

The result will look as follows:

<user>admin</admin>
<error>Integer conversion failed: abc</error>

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

Updates

The @Updating annotation can be applied to mark Java functions that perform write or update operations:

    @Updating
    public void backup() { 
        // ...
    }

An XQuery expression will be handled as an updating expression if it calls an updating Java function. In contrast to XQuery update operations, the Java code will immediately be executed, but the result will be cached as if update:output was called.

The annotation is particularly helpful if combined with a lock annotation.
Locking

By default, a Java function will be executed in parallel with other code. If a Java function performs sensitive operations, it is advisable to explicitly lock the code.

Java Locks

Java provides a handful of mechanism to control the execution of code. The concurrent execution of functions can be avoided with the `synchronized` keyword. For more complex scenarios, the Lock, Semaphore and Atomic classes can be brought into play.

XQuery Locks

If you want to synchronize the execution of your code with BaseX locks, you can take advantage of the `@Lock` annotation:

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

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

If an XQuery expression invokes `write()`, any other query that calls `write()` or `read()` needs to wait for the query to be finished. The `read()` function can be run in parallel; whereas queries will be queued if `write()` is called.

More details on concurrent querying can be found in the article on Transaction Management.

Data Types

Conversion to Java

Before Java code is executed, the arguments are converted to Java values, depending on the addressed function or constructor parameters. The accepted Java types and the original XQuery types are depicted in the second and first column of the table below.

If a numeric value is supplied for which no exact matching is defined, it is cast to the appropriate type unless it exceeds its limits. The following two function calls are equivalent:

```java
Q{String}codePointAt('ABC', xs:int(1)),
Q{String}codePointAt('ABC', xs:byte(1)),
Q{String}codePointAt('ABC', 1)
```

Conversion to XQuery

By default, Java values with the most common types (as shown in the second and third column of the table) are converted to XQuery values. All other values are returned as Java items, which are function items with a wrapped Java value. The results of constructor calls are always returned as Java items.

The conversion of the wrapped Java value to XQuery is enforced by invoking the function item: Values in `Iterator` and `Iterable` instances (Lists, Sets and Collections) are converted to items, and maps are converted to XQuery maps:

```java
declare namespace Scanner = 'java:java.util.Scanner';
```
let $scanner := Scanner:new("A B C") => Scanner:useDelimiter("\n")
return $scanner()

If no conversion is defined, a string is returned, resulting from the toString() method of the object. This method is also called if the string representation of a Java item is requested:

!(: returns the string representations of a HashMap and an ArrayList instance :) 'Map: ' || Q{java.util.HashMap}new(), string(Q{java:java.util.ArrayList}new())

The conversion can be further controlled with the WRAPJAVA option. The following values exist:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>some</td>
<td>The default: Java values of the most common types are converted, others are wrapped into Java items.</td>
</tr>
<tr>
<td>none</td>
<td>All Java values are converted. If no conversion is defined, a string is returned, resulting from the toString() method.</td>
</tr>
<tr>
<td>all</td>
<td>Java values are wrapped into Java items (excluding those inheriting the internal type org.basex.query.value.Value).</td>
</tr>
<tr>
<td>instance</td>
<td>If the method of a class instance was called, the Java value is ignored and the instance is wrapped into a Java item. Otherwise, the Java value is returned.</td>
</tr>
<tr>
<td>void</td>
<td>Java values are ignored, and an empty sequence is returned instead.</td>
</tr>
</tbody>
</table>

In the following example, the result of the first function – a char array – is wrapped and passed on to a CharBuffer function. Without the option, the single-value array would be converted to an xs:unsignedShort item and the second function call would fail:

!(: Without the pragma, the result of toChars would be converted to an xs:unsignedShort item, and the second function call would fail :) #)

Q{Character}toChars(xs:int(33)) => Q{java.nio.CharBuffer}wrap()

The next example demonstrates a use case for the instance option:

!(: Thanks to the pragma, the function calls can be chained :) #)
declare namespace set = 'java:java.util.HashSet';
let $set := (# db:wrapjava instance #) {
  set:new()
  => set:add('1')
  => set:add('2')
}
return $set()

The void option is helpful if side-effecting methods return values that do not contribute to the final result:

!(: Without the pragma, 100 booleans would be returned by the FLWOR expression :) #)
declare namespace set = 'java:java.util.HashSet';
let $set := set:new()
return (
  (# db:wrapjava void #) {
    for $i in 1 to 100
    return set:add($set, $i)
  }
)
The irrelevant results could also be swallowed with `prof: void`.

<table>
<thead>
<tr>
<th>XQuery input</th>
<th>Expected or returned Java type</th>
<th>XQuery output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>item()</code>* (no conversion)</td>
<td>org.basex.query.value.Value</td>
<td><code>item()</code>* (no conversion)</td>
</tr>
<tr>
<td><code>empty-sequence()</code></td>
<td>null</td>
<td><code>empty-sequence()</code></td>
</tr>
<tr>
<td><code>xs:string</code></td>
<td>String</td>
<td><code>xs:string</code></td>
</tr>
<tr>
<td><code>xs:unsignedAtomic</code></td>
<td>char, Character</td>
<td><code>xs:unsignedShort</code></td>
</tr>
<tr>
<td><code>xs:boolean</code></td>
<td>boolean, Boolean</td>
<td><code>xs:boolean</code></td>
</tr>
<tr>
<td><code>xs:byte</code></td>
<td>byte, Byte</td>
<td><code>xs:byte</code></td>
</tr>
<tr>
<td><code>xs:short</code></td>
<td>short, Short</td>
<td><code>xs:short</code></td>
</tr>
<tr>
<td><code>xs:int</code></td>
<td>int, Integer</td>
<td><code>xs:int</code></td>
</tr>
<tr>
<td><code>xs:integer, xs:long</code></td>
<td>long, Long</td>
<td><code>xs:integer</code></td>
</tr>
<tr>
<td><code>xs:unsignedLong</code></td>
<td>java.math.BigInteger</td>
<td><code>xs:unsignedLong (lossy)</code></td>
</tr>
<tr>
<td><code>xs:decimal</code></td>
<td>java.math.BigDecimal</td>
<td><code>xs:decimal</code></td>
</tr>
<tr>
<td><code>xs:float</code></td>
<td>float, Float</td>
<td><code>xs:float</code></td>
</tr>
<tr>
<td><code>xs:double</code></td>
<td>double, Double</td>
<td><code>xs:double</code></td>
</tr>
<tr>
<td><code>xs:QName</code></td>
<td>javax.xml.namespace.QName</td>
<td><code>xs:QName</code></td>
</tr>
<tr>
<td><code>xs:anyURI</code></td>
<td>java.net.URI, java.net.URL</td>
<td><code>xs:anyURI</code></td>
</tr>
<tr>
<td><code>xs:duration</code></td>
<td>javax.xml.datatype.XMLGregorianCalendar</td>
<td><code>xs:duration</code></td>
</tr>
<tr>
<td><code>node()</code></td>
<td>org.w3c.dom.Node</td>
<td><code>node()</code></td>
</tr>
<tr>
<td><code>array(xs:boolean)</code></td>
<td>boolean[]</td>
<td><code>xs:boolean*</code></td>
</tr>
<tr>
<td><code>array(xs:string)</code></td>
<td>String[]</td>
<td><code>xs:string*</code></td>
</tr>
<tr>
<td><code>array(xs:unsignedShort)</code></td>
<td>char[]</td>
<td><code>xs:unsignedShort*</code></td>
</tr>
<tr>
<td><code>array(xs:short)</code></td>
<td>short[]</td>
<td><code>xs:short*</code></td>
</tr>
<tr>
<td><code>array(xs:int)</code></td>
<td>int[]</td>
<td><code>xs:int*</code></td>
</tr>
<tr>
<td><code>array(xs:integer), array(xs:long)</code></td>
<td>long[]</td>
<td><code>xs:integer*</code></td>
</tr>
<tr>
<td><code>array(xs:float)</code></td>
<td>float[]</td>
<td><code>xs:float*</code></td>
</tr>
<tr>
<td><code>array(xs:double)</code></td>
<td>double[]</td>
<td><code>xs:double*</code></td>
</tr>
<tr>
<td><code>Object[] (others)</code></td>
<td>item()*</td>
<td><code>array(*)(others)</code></td>
</tr>
<tr>
<td><code>map(*)</code></td>
<td>java.util.HashMap</td>
<td>Wrapped Java object</td>
</tr>
</tbody>
</table>

**URI Rewriting**

Before a Java class or module is accessed, its namespace URI will be normalized:

1. If the URI is a URL:
   a. 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. Otherwise, if the URI is a URN, colons will be replaced with slashes.

3. Characters other than letters, dots and slashes will be replaced with dashes.

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

If the resulting path has no file suffix, it may point to either an XQuery module or a Java archive:

- 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 9.6

- Updated: Java Bindings revised (new mappings, Java function items, WRAPJAVA option).

Version 9.4

- Added: Annotation for updating functions.
- Updated: Single annotation for read and write locks.

Version 8.4

- Updated: Rewriting rules

Version 8.2

- Added: URI Rewriting: support for URNs

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
- Added: Packaging, URI Rewriting
Chapter 28. 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 things that makes languages successful is the availability of external libraries. As XQuery comes with only 150 pre-defined functions, which cannot meet all requirements, additional library modules exist – such as FunctX – which extend the language with new features.

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

1. The internal Packaging mechanism will install single XQuery and JAR 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** hello-universe.xq:

```xquery
import module namespace m = 'http://basex.org/modules/hello' at 'hello-world.xqm';
mihello("Universe")
```

**Library Module** hello-world.xqm (in the same directory):

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

If no location is supplied, modules will be looked up in the repository. Repository modules are stored in the repo directory, which resides in your home directory. XQuery modules can be manually copied to the repository directory or installed and deleted via commands.

The following example calls a function from the FunctX module in the repository:

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

Commands

There are various ways to organize 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 BaseX.
Installation

A module or package can be installed with REPO INSTALL. The path to the file has to be given as a parameter:

```
REPO INSTALL https://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 REPO LIST. The names of all packages are listed, along with their version, their package type, and the repository path:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.functx.com">http://www.functx.com</a></td>
<td>1.0</td>
<td>EXPath</td>
<td>http-www.functx.com-1.0</td>
</tr>
</tbody>
</table>
```

Removal

A package can be deleted with 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
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 the file can successfully be parsed, the module URI will be rewritten to a file path and attached with the .xqm file suffix, and the original file will possibly be renamed and copied to that path into the repository.

Example:

Installation (the original file will be copied to the org/basex/modules/Hello subdirectory of the repository):

```
REPO INSTALL https://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

For general notes on importing Java classes, please read the Java Bindings article on Module Imports.

Java archives (JARs) 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 the .jar file suffix, and the original file will be renamed and copied to that path into the repository.

If the class will be imported in the prolog of the XQuery module, an instance of it will be created, and its public functions can then be addressed from XQuery. A class may extend the QueryModule class to get access to the current query context and to be enriched by some helpful annotations (see Annotations).
Example:

Structure of the HelloWorld.jar archive:

```
META-INFO/
   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 having installed the module, all of the following URIs can be used in XQuery to import this module or call its functions (see URI Rewriting for more information):

```
http://basex.org/modules/Hello
org/basex/modules/Hello
org.basex.modules.Hello
```

Additional Libraries

A Java class may depend on additional libraries. The dependencies can be resolved by creating a fat JAR file, i.e., extracting all files of the library archives and producing a single, flat JAR package.

Another solution is to copy the libraries into a lib directory of the JAR package. When the package is installed, the additional library archives will be extracted and copied to a hidden subdirectory in the repository. If the package is deleted, the hidden subdirectory will be removed as well.

Exampary contents of Image.jar

```
lib/
   Images.jar
META-INF/
   MANIFEST.MF
org/basex/modules/
   Image.class
```

Directory structure of the repository directory after installing the package:

```
org/basex/modules/
   Image.class
```
It makes sense to combine the advantages of XQuery and Java packages:

- Instead of directly calling Java code, a wrapper module can be provided. This module contains functions that invoke the Java functions.
- These functions can be strictly typed. This reduces the danger of erroneous or unexpected conversions between XQuery and Java code.
- In addition, the entry functions can have properly maintained XQuery comments.

XQuery and Java can be combined as follows:

- First, a JAR package is created (as described above).
- A new XQuery wrapper module is created, which is named identically to the Java main class.
- The URL of the import module statement in the wrapper module must start with the java: prefix.
- The finalized XQuery module must be copied into the JAR file, and placed in the same directory as the Java main class.

If the resulting JAR file is installed, the embedded XQuery module will be extracted, and will be called first if the module will be imported.

**Main Module** hello-universe.xq

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

**Wrapper Module** Hello.xqm

```xquery
module namespace hello = 'http://basex.org/modules/Hello';

(: Import JAR file :) 
import module namespace java = 'java:org.basex.modules.Hello';

(:~
    : Say hello to someone.
    : @param  $world  the one to be greeted
    : @return welcome string
    :
) declare function hello:hello(
    $world  as xs:string
) as xs:string {
    java:hello($world)
};
```

**Java class** Hello.java

```java
package org.basex.modules;

public class Hello {
    public String hello(final String world) {
        return "Hello " + world;
    }
}
```

If the JAR file is installed, **Combined** will be displayed as type:
EXPath Packaging

The EXPath specification defines the structure of a .xar archive. The package contains at its root a package descriptor named expath-pkg.xml. This descriptor presents some metadata 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

If you want to package an EXPath archive with Java code, 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.

**Performance**

Importing XQuery modules that are located in the repository is just as fast as importing any other modules. Modules that are imported several times in a project will only be compiled once.

Imported Java archives will be dynamically added to the classpath and unregistered after query execution. This requires some constant overhead and may lead to unexpected effects in scenarios with highly concurrent read operations. If you want to get optimal performance, it is recommendable to move your JAR files into the lib/custom directory of BaseX. This way, the archive will be added to the classpath if BaseX is started. If you have installed a Combined Package, you can simply keep your XQuery module in the repository, and the Java classes will be automatically detected.

**Changelog**

Version 9.0

- Added: Combined XQuery and Java packages
- Added: Additional Libraries

Version 7.2.1

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

Version 7.1

- Added: Repository Module

Version 7.0

- Added: EXPath Packaging
Chapter 29. Full-Text

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It summarizes the features of the W3C XQuery Full Text Recommendation, and custom features of the implementation in BaseX.

Please read the separate Full-Text Index section in our documentation if you want to learn how to evaluate full-text requests on large databases within milliseconds.

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. BaseX was the first query processor that supported all features of the specification.

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

Various 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 f tand, ftor and fnot can also be used to combine multiple query terms. The following query yields the same result as the last one does:

```
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:

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

Due to the complex data model of the XQuery Full Text spec, the usage of `ftand` may lead to a high memory consumption. If you should encounter problems, simply use the `all` keyword:

```xml
doc('factbook')//country[descendant::religions contains text { 'Christian', 'Jewish'} all]/name
```

### Positional Filters

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

```xml
<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?

```xml
("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:

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

By the way: In some examples above, the `words` unit was used, but `sentences` and `paragraphs` would have been 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" using diacritics sensitive

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

"catch" contains text "catches" using stemming

• With the stop words option, a list of words can be defined that will be ignored when tokenizing a string. This is particularly helpful if the full-text index takes too much space (a standard stopword list for English texts is provided in the directory etc/stopwords.txt in the full distributions of BaseX, and available online at http://files.basex.org/etc/stopwords.txt):

"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**

**Languages**

The chosen language determines how strings will be tokenized and stemmed. Either names (e.g. English, German) or codes (en, de) can be specified. A list of all language codes that are available on your system can be retrieved as follows:

```xml
declare namespace locale = "java:java.util.Locale";
distinct-values(locale:getAvailableLocales() ! locale:getLanguage(.))
```

By default, unless the languages codes ja, ar, ko, th, or zh are specified, a tokenizer for Western texts is used:

• Whitespaces are interpreted as token delimiters.

• Sentence delimiters are .. !, and ?.

• Paragraph delimiters are newlines (\n\n).

The basic JAR file of BaseX comes with built-in stemming support for English, German, Greek and Indonesian. Some more languages are supported if the following libraries are found in the classpath:

• lucene-stemmers-3.4.0.jar includes the Snowball and Lucene stemmers for the following languages: Arabic, Bulgarian, Catalan, Czech, Danish, Dutch, Finnish, French, Hindi, Hungarian, Italian, Latvian, Lithuanian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish, Turkish.

• igo-0.4.3.jar : An additional article explains how Igo can be integrated, and how Japanese texts are tokenized and stemmed.
The JAR files are included in the ZIP and EXE distributions 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 "German"
```

**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:

```xquery
let $terms := ('a', 'b')
let $scores := ft:score($terms ! ('a b c' contains text { . } ))
return avg($scores)
```

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

Scoring values can be further processed to compute custom values:

```xquery
let $string := 'a b'
return ft:score($string contains text 'a' ftand 'b'),
```

Scoring is supported within full-text expressions, by `ft:search`, and by simple predicate tests that can be rewritten to `ft:search`:

```xquery
for $n score $s in ft:search('factbook', 'orthodox')
order by $s descending
return $s || ': ' || $n,
```

For a thesaurus file, the following query returns `false`:

```
'hardware' contains text 'computers'
```

If a thesaurus is employed...

```xquery
<thesaurus xmlns="http://www.w3.org/2007/xqftts/thesaurus">
  <entry>
    <term>computers</term>
    <synonym>
      <term>hardware</term>
      <relationship>NT</relationship>
    </synonym>
  </entry>
</thesaurus>
```

...the result will be `true`:
'hardware' contains text 'computers'
using thesaurus at 'thesaurus.xml'

Thesaurus files must comply with the XSD Schema of the XQFT Test Suite (but the namespace can be omitted). Apart from the relationship defined in ISO 2788 (NT: narrower term, RT: related term, etc.), custom relationships can be used.

The type of relationship and the level depth can be specified as well:

((: BT: find broader terms; NT means narrower term :)
'computers' contains text 'hardware'
using thesaurus at 'x.xml' relationship 'BT' from 1 to 10 levels

More details can be found in the specification.

**Fuzzy Querying**

In addition to the official recommendation, BaseX supports a fuzzy search feature. The XQFT grammar was enhanced by the *fuzzy* match option to allow for approximate results in full texts:

**Document 'doc.xml':**

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

**Query:**

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

**Result:**

```xml
<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. 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”.

A user-defined value can be adjusted globally via the **LSERROR** option or via an additional argument:

```xml
//a[text() contains text 'house' using fuzzy 3 errors]
```

**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:

```xml
<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, it is recommendable to:

- Keep *whitespace stripping* turned off when importing XML documents. This can be done by ensuring that **STRIPWS** is disabled. This can also be done in the GUI if a new database is created (Database → New... → Parsing → Strip Whitespaces).
- Keep automatic indentation turned off. Ensure that the **serialization parameter indent** is set to no.
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 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 will only yield useful results if they are applied to single text nodes, as the following example demonstrates:

```xml
(ft:mark('//p[contains(text(), 'real')]'))
(ft:mark('//p[text()] contains text('real')]'))
```

BaseX does not support the `ignore option` (without `content`) of the W3C XQuery Full Text 1.0 Recommendation. If you want to ignore descendant element content, such as footnotes or other material that does not belong to the same logical text flow, you can build a second database from and exclude all information you want to avoid searching for. See the following example (visit XQuery Update to learn more about updates):

```xml
let $docs := db:get('docs')
return db:create('index-db',
    $docs update delete node (
        ./footnote
    ),
    $docs/db:path(.),
    map { 'ftindex': true() }
)
```

## 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:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lang</td>
<td>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: <code>de</code>, <code>en-US</code></td>
</tr>
<tr>
<td>strength</td>
<td>Level of difference considered significant in comparisons. Four strengths are supported: primary, secondary, tertiary, and identical. As an example, in German:</td>
</tr>
<tr>
<td></td>
<td>- “Ä” and “A” are considered primary differences,</td>
</tr>
<tr>
<td></td>
<td>- “Ä” and “ü” are secondary differences,</td>
</tr>
<tr>
<td></td>
<td>- “Ä” and “AĴ” (see <a href="http://www.fileformat.info/info/unicode/char/308/index.htm">http://www.fileformat.info/info/unicode/char/308/index.htm</a>) are tertiary differences, and</td>
</tr>
</tbody>
</table>
• "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:

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

If the ICU Library is added to the classpath, the full Unicode Collation Algorithm features become available:

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

Changelog

Version 9.6

• Updated: Fuzzy Querying: Specify Levenshtein error

Version 9.5

• Removed: Scoring propagation.

Version 9.2

• Added: Arabic stemmer.

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 30. 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, the challenges are addressed that arise due to the functional semantics 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 //n

The example query deletes all <n> 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 //originalNode
return rename node $n as 'renamedNode'

All originalNode 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.
Main-Memory Updates

copy/modify/return

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

The originalNode 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:get('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

Updated with Version 10: Curly braces are now mandatory.

The update expression is a BaseX-specific convenience operator for the copy/modify/return construct:

• Similar to the XQuery 3.0 map operator, the value of the first
XQuery Update

expression is bound as context item, and the second expression performs updates on this item. The updated item is returned as result:

```xml
for $item in db:get('data')//item
return $item update {
    delete node text()
}
```

- More than one node can be specified as source:

```xml
db:get('data')//item update {
    delete node text()
}
```

- If wrapped with curly braces, update expressions can be chained:

```xml
<root/> update {
    insert node <child/> into .
} update {
    insert node "text" into child
}
```

**transform with**

The `transform with` expression was added to the current XQuery Update 3.0 working draft. It is a simple version of the `update` expression and also available in BaseX:

```xml
<xml>text</xml> transform with {
    replace value of node . with 'new-text'
}
```

**Functions**

**Built-in Functions**

`fn:put()` can be used to serialize XDM instances to secondary storage:

- The function will be executed after all other updates.
- Serialized documents therefore reflect all changes made effective during a query.
- No files will be created if the addressed nodes have been deleted.
- Serialization parameters can be specified as third argument (more details are found in the [XQUF 3.0 Specification](#)).

Numerous additional database functions exist for performing updates on document and database level.

**User-Defined Functions**

If an updating function item is called, the function call must be prefixed with the keyword `updating`. This ensures that the query compiler can statically detect if an invoked function item will perform updates or not:

```xml
let $node := <node>TO-BE-DELETED</node>
let $delete-text := %updating function($node) {
    delete node $node//*[text()]
}
return $node update {
    updating $delete-text(.)
}
```

As shown in the example, user-defined and anonymous functions can additionally be annotated as `%updating`. 
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

Updated with Version 10: db:put-binary is executed before standard XQuery Update expressions.

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

- **Backups, Binary resources**: db:alter-backup, db:create-backup, db:put, db:put-binary
- **XQuery Update**: insert before, delete, replace, rename, replace value, insert attribute, insert into first, insert into, insert into last, insert, insert after, fn:put
- **Documents**: db:add, db:put, 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**: 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 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 root expression of a query must be a sequence of updating expressions. But there are two ways out:
• The BaseX-specific `update: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:

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

• With `MIXUPDATES`, 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.

**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 if `WRITEBACK` 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.

Please remember that the collected updates will be executed after the query evaluation. All logical errors will be raised before the updates are actually executed.

**Changelog**

Version 10.0

• Updated: `db:put-binary` is executed before XQuery Update expressions.

• Updated: `update`: Curly braces are now mandatory.

Version 9.0

• Updated: `Built-in Functions`: serialization parameters

Version 8.5

• Added: `transform with`

• Updated: `update` was extended.

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 31. Indexes

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal. It contains information on the available index structures.

The query compiler tries to optimize and speed up queries by applying the index whenever it is possible, and seems promising. To see how a query is rewritten, and if an index is used, you can turn on the Info View in the GUI or use the -V flag on the command line:

- A message like apply text index for "Japan" indicates that the text index is applied to speed up the search of the shown string. The following message…

- no index results indicates that a string in a path expression will never yield results. Hence, the path does not need to be evaluated at all.

- If you cannot find any index optimization hints in the info output, it often helps if you rewrite and simplify your query.

Additional examples for index rewritings are presented in our article on XQuery Optimizations.

Structural Indexes

Structural indexes are automatically created and cannot be dropped by the user:

Name Index

The name index contains references to the names of all elements and attributes in a database. It contains some basic statistical information, such as the number of occurrence of a name.

The name index is e.g. applied to discard location steps that will never yield results:

```xml
(: 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.

If a database is updated, new names will be added incrementally, but the statistical information will get out-dated.

Path Index

The path index (which is also called path summary or data guide) stores all distinct paths of the documents in the database. It contains additional statistical information, such as the number of occurrence of a path, its distinct string values, and the minimum/maximum of numeric values. The maximum number of distinct values to store per name can be changed via MAXCATS. Distinct values are also stored for elements and attributes of numeric type.

Various queries will be evaluated much faster if an up-to-date path index is available (as can be observed when opening the Info View):

- Descendant steps will be rewritten to multiple child steps. Child steps are evaluated faster, as fewer nodes have to be traversed:

```xml
(doc('factbook.xml')//province,
 (: ...will be rewritten to... :) 
doc('factbook.xml')/mondial/country/province
```

- The fn:count function will be pre-evaluated by looking up the number in the index:

```xml
count(doc('factbook')//country)
```
Indexes

- The distinct values of elements or attributes can be looked up in the index as well:

```xquery
distinct-values(db:get('factbook')//religions)
```

The contents of the path index can be directly accessed with the XQuery function `index:facets`.

If a database is updated, the statistics in the path index will be invalidated.

**Document Index**

The document index contains references to all document nodes in a database. Once documents with specific paths are requested, the index will be extended to also contain document paths.

The index generally speeds up access to single documents and database paths. It will always be kept up-to-date.

**Value Indexes**

Value indexes can be created and dropped by the user. Four types of value indexes are available: a text and attribute index, and an optional token and full-text index. By default, the text, and attribute index will automatically be created.

In the GUI, index structures can be managed in the dialog windows for creating new databases or displaying the database properties. On command-line, the commands `CREATE INDEX` and `DROP INDEX` are used to create and drop index structures. With `INFO INDEX`, you get some insight into the contents of an index structure, and `SET` allows you to change the index defaults for new databases:

- `OPEN factbook; CREATE INDEX fulltext`: Open database; create full-text index
- `OPEN factbook; INFO INDEX TOKEN`: Open database; show info on token index
- `SET ATTRINDEX true; SET ATTRINCLUDE id name; CREATE DB factbook.xml`: Enable attribute index; only index 'id' and 'name' attributes; create database

With XQuery, index structures can be created and dropped via `db:optimize`:

```xquery
(db:optimize('factbook', false(), map { 'ftindex': true(), 'ftinclude': 'p div' })
```

**Text Index**

**Exact Queries**

This index references text nodes of documents. It will be utilized to accelerate string comparisons in path expressions. The following queries will all be rewritten for index access:

```xquery
//*[text() = 'Germany'],
(doc('factbook.xml')//name[. = 'Germany'],
for $c in db:get('factbook')//country
where $c//city/name = 'Hanoi'
return $c/name
```

Before the actual index rewriting takes places, some preliminary optimizations are applied:

- In example 2, the context item expression `. = 'Germany'` will be replaced with a `text()` step.
Indexes

• In example 3, the `where` clause will be rewritten to a predicate and attached to the first path expression.

The indexed text nodes can be accessed directly with the XQuery function `db:text`. The indexed string values can be looked up via `index:text`.

The **UPDINDEX** option can be enabled to keep this index up-to-date:

```xml
db:optimize(
    'mydb',
    true(),
    map { 'updindex':true(), 'textindex': true(), 'textinclude':'id' }
)
```

### Range Queries

The text index also supports range queries based on string comparisons:

1. `db:get('Library')//Medium[Year >= '2011' and Year <= '2016']`,
2. let $min := '2014-04-16T00:00:00' 
   return db:get('news')//entry[date-time > $min and date-time < $max]

With `db:text-range`, you can access all text nodes whose values are between a minimum and maximum value.

Please note that the index structures do not support queries for numbers and dates.

### Attribute Index

Similar to the text index, this index speeds up string and range comparisons on attribute values. Additionally, the XQuery function `fn:id` takes advantage of the index whenever possible. The following queries will all be rewritten for index access:

1. `//country[@car_code = 'J']`,
2. `//province[@* = 'Hokkaido']//name`,
3. `//sea[@depth > '2100' and @depth < '4000']`
4. `fn:id('f0_119', db:get('factbook'))`

**Attribute nodes** (which you can use as starting points of navigation) can directly be retrieved from the index with the XQuery functions `db:attribute` and `db:attribute-range`. The index contents (**strings**) can be accessed with `index:attributes`.

The **UPDINDEX** option can be activated to keep this index up-to-date.

### Token Index

In many XML dialects, such as HTML or DITA, multiple tokens are stored in attribute values. The token index can be created to speed up the retrieval of these tokens. The XQuery functions `fn:contains-token`, `fn:tokenize` and `fn:idref` are rewritten for index access whenever possible. If a token index exists, it will, e.g., be utilized for the following queries:

1. `//div[contains-token(@class, 'row')]`,
2. `//p[tokenize(@class) = 'row']`,
3. `doc('graph.xml')/idref('edge8')`
Indexes

Attribute nodes with a matching value (containing at least one from a set of given tokens) can be directly retrieved from the index with the XQuery function `db:token`. The index contents (token strings) can be accessed with `index:tokens`.

Full-Text Index

The Full-Text index contains the normalized tokens of text nodes of a document. It is utilized to speed up queries with the `contains text` expression, and it is capable of processing wildcard and fuzzy search operations. 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).

If the full-text index exists, the following queries will all be rewritten for index access:

```
//country[name/text() contains text 'and'],
//religions[]//text() contains text { 'Catholic', 'Roman' } using case insensitive distance at most 2 words
```

The index provides support for the following full-text features (the values can be changed in the GUI or via the SET command):

- **Stemming**: tokens are stemmed before being indexed (option: STEMMING)
- **Case Sensitive**: tokens are indexed in case-sensitive mode (option: CASESENS)
- **Diacritics**: diacritics are indexed as well (option: DIACRITICS)
- **Stopword List**: a stop word list can be defined to reduce the number of indexed tokens (option: STOPWORDS)
- **Language**: see Languages for more details (option: LANGUAGE)

The options that have been used for creating the full-text index will also be applied to the optimized full-text queries. However, the defaults can be overwritten if you supply options in your query. For example, if words were stemmed in the index, and if the query can be rewritten for index access, the query terms will be stemmed as well, unless stemming is not explicitly disabled. This is demonstrated in the following Command Script:

```
<commands>
  <!-- Create database with stemmed full-text index -->
  <set option='stemming'>true</set>
  <create-db name='test-db'> <text>house</text> </create-db>
  <!-- Index access: Query term will be stemmed -->
  <xquery> /text[. contains text { 'houses' }] </xquery>
  <!-- Disable stemming (query will not be evaluated by the index) -->
  <xquery> /text[. contains text { 'houses' } using no stemming] </xquery>
</commands>
```

Text nodes can be directly requested from the index via the XQuery function `ft:search`. The index contents can be accessed with `ft:tokens`.

Selective Indexing

Value indexing can be restricted to specific elements and attributes. The nodes to be indexed can be restricted via the `TEXTINCLUDE`, `ATTRINCLUDE`, `TOKENINCLUDE` and `FTINCLUDE` options. The options take a list of name patterns, which are separated by commas. The following name patterns are supported:

- `*` : all names
- `name` : elements or attributes called `name`, which are in the empty default namespace
- `*:name` : elements or attributes called `name`, no matter which namespace
Indexes

- Q{uri}*: all elements or attributes in the uri namespace
- Q{uri}name*: elements or attributes called name in the uri namespace

The options can either be specified via the SET command or via XQuery. With the following operations, an attribute index is created for all id and name attributes:

**Commands**

```
SET ATTRINCLUDE id,name
CREATE DB factbook http://files.basex.org/xml/factbook.xml'
# Restore default
SET ATTRINCLUDE
```

**XQuery**

```
db:create('factbook', 'http://files.basex.org/xml/factbook.xml', '',
    map { 'attrinclude': 'id,name' })
```

With CREATE INDEX and db:optimize, new selective indexing options will be applied to an existing database.

**Enforce Rewritings**

In various cases, existing index structures will not be utilized by the query optimizer. This is usually the case if the name of the database is not a static string (e.g. because it is bound to a variable or passed on as an argument of a function call). Furthermore, several candidates for index rewritings may exist, and the query optimizer may decide for a rewriting that turns out to be suboptimal.

With the ENFORCEINDEX option, certain index rewritings can be enforced. While the option can be globally enabled, it is usually better to supply it as Pragma. Two examples:

- In the query below, 10 databases will be addressed. If it is known in advance that these databases contain an up-to-date text index, the index rewriting can be enforced as follows:

  ```
  (# db:enforceindex #) {
  for $n in 1 to 10
  let $db := 'persons' || $n
  return db:get($db)//person[name/text() = 'John']
  }
  ```

- The following query contains two predicates that may both be rewritten for index access. If the automatically chosen rewriting is known not to be optimal, another index rewriting can enforced by surrounding the specific expression with the pragma:

  ```
  db:get('factbook')//country
  [(# db:enforceindex #) {
  @population > '10000000' and
  @population < '10999999'
  }]
  [religions/text() = 'Protestant']
  ```

The option can also be assigned to predicates with dynamic values. In the following example, the comparison of the first comparison will be rewritten for index access. Without the pragma expression, the second comparison is preferred and chosen for the rewriting because the statically known string allows for an exact cost estimation:

```
for $name in ('Germany', 'Italy')
for $country in db:get('factbook')//country
where (# db:enforceindex #) { $country/name = $name }
where $country/religions/text() = 'Protestant'
return $country
```

Please note that:
Indexes

- The option should only be enabled if the addressed databases exist, have all required index structures and are up-to-date (otherwise, you will be given an error message).

- If you address the full-text index, and if you use non-default indexing options, you will have to specify them in your query (via using stemming, using language 'de', etc).

- If you have more than one enforce pragma in a single path expression, only the first will be considered.

- In general, there are always expressions that cannot be rewritten for index access. If you enforce rewritings, you will have no guarantee that an index will be used.

Custom Index Structures

With XQuery, it is comparatively easy to create your own, custom index structures. The following query demonstrates how you can create a factbook-index database, which contains all texts of the original database in lower case:

```xml
let $db := 'factbook'
let $index := <index>{
  for $nodes in db:get($db)//text() 
group by $text := lower-case($nodes) 
return <text string='{ $text }'>{
  for $node in $nodes 
  return <id>{ db:node-id($node ) } </id>
  </text>
}</index>
return db:create($db || '-index', $index, $db || '-index.xml')
```

In the following query, a text string is searched, and the text nodes of the original database are retrieved:

```xml
let $db := 'factbook'
let $text := 'italian'
for $id in db:get($db || '-index')//*[@string = $text]/id 
return db:get-id($db, $id)/..
```

With some extra effort, and if UPDINDEX is enabled for both your original and your index database (see below), your index database will support updates as well (try it, it’s fun!).

Performance

If main memory runs out while creating a value index, the current index structures will be partially written to disk and eventually merged. If the memory heuristics fail for some reason (i.e. because multiple index operations run at the same time, or because the applied JVM does not support explicit garbage collections), a fixed index split sizes may be chosen via the SPLITSIZE option.

If DEBUG is enabled, the command-line output might help you find a good split size. The following example shows the output for creating a database for an XMark document with 1 GB, and with 128 MB assigned to the JVM:

```
> basex -d -c"SET FTINDEX ON; SET TOKENINDEX ON; CREATE DB xmark 1gb.xml"
Creating Database...
................................ 76559.99 ms (29001 KB)
Indexing Text...
....|...|...|.....|. 9.81 M operations, 18576.92 ms (13523 KB). Recommended SPLITSIZE: 20.
Indexing Attribute Values...
........|........ 3.82 M operations, 7151.77 ms (6435 KB). Recommended SPLITSIZE: 20.
Indexing Tokens...
........|........ 3.82 M operations, 9636.73 ms (10809 KB). Recommended SPLITSIZE: 10.
```
Indexes

Indexing Full-Text...

116.33 M operations, 138740.94 ms (106 MB). Recommended SPLITSIZE: 12.

The output can be interpreted as follows:

• The vertical bar | indicates that a partial index structure was written to disk.

• The mean value of the recommendations can be assigned to the SPLITSIZE option. Please note that the recommendation is only a vague proposal, so try different values if you get main-of-memory errors or indexing gets too slow. Greater values will require more main memory.

• In the example, the full-text index was split 12 times. 116 million tokens were indexed, processing time was 2.5 minutes, and final main memory consumption (after writing the index to disk) was 76 MB. A good value for the split size option could be 15.

Updates

Generally, update operations are very fast in BaseX. By default, the index structures will be invalidated by updates; as a result, queries that benefit from index structures may slow down after updates. There are different alternatives to cope with this:

• After the execution of one or more update operations, the OPTIMIZE command or the db:optimize function can be called to rebuild the index structures.

• The UPDINDEX option can be activated before creating or optimizing the database. As a result, the text, attribute and token indexes will be incrementally updated after each database update. Please note that incremental updates are not available for the full-text index and database statistics. This also explains why the UPTODATE flag, which is e.g. displayed via INFO DB or db:info, will be set to false until the database will be optimized again (various optimizations won’t be triggered. For example, count(/item) can be extremely fast if all metadata is up-to-date.

• The AUTOOPTIMIZE option can be enabled before creating or optimizing the database. All outdated index structures and statistics will then be recreated after each database update. This option should only be done for small and medium-sized databases.

• Both options can be used side by side: UPDINDEX will take care that the value index structures will be updated as part of the actual update operation. AUTOOPTIMIZE will update the remaining data structures (full-text index, database statistics).

Changelog

Version 9.1

• Updated: Enforce Rewritings, support for comparisons with dynamic values.

Version 9.0

• Added: Enforce Rewritings

Version 8.4

• Updated: Name Index, Path Index

Version 8.4

• Added: Token Index

Version 8.3

• Added: Selective Indexing
Indexes

Version 8.0
• Added: AUTOOPTIMIZE option

Version 7.2.1
• Added: string-based range queries
Chapter 32. 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 will be serialized (i.e., returned to the client or an API, usually in textual form). The official parameters are defined in the W3C XQuery Serialization 3.1 document. In BaseX, they can be:

- included in the prolog of the XQuery expression;
- specified in XQuery functions (`file:write`, `db:export`, `fn:serialize()`);
- specified in REST query parameters;
- specified in RESTXQ output annotations;
- set via the `SERIALIZER` option before running a query;
- set via the `EXPORTER` option before exporting a database; or
- supplied with the `-s` flag of the BaseX command-line clients.

The namespace for serialization parameters is statically bound to the `output` prefix. This means that it need not (but may) be declared in the query prolog:

```xml
declare namespace output = 'http://www.w3.org/2010/xslt-xquery-serialization';
declare option output:method 'text';
<xml>Hi there</xml>
```

Due to the wide range of ways how parameters can be supplied, we deliberately ignored one rule of the specification, which requires non-official features to be defined in a non-null namespace URI. In the following, we will indicate which features are specific to our implementation.

**Parameters**

The following serialization parameters are supported by BaseX (further details can be looked up in the official specification):

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>Specifies the serialization method. <code>xml</code>, <code>xhtml</code>, <code>html</code>, <code>text</code> and <code>adaptive</code> are part of the official specification. For more details on <code>basex</code>, <code>csv</code> and <code>json</code>, see <a href="#">XQuery Extensions</a>.</td>
<td><code>xml</code>, <code>xhtml</code>, <code>html</code>, <code>text</code>, <code>json</code>, <code>adaptive</code>, <code>csv</code>, <code>basex</code></td>
<td><code>basex</code></td>
</tr>
<tr>
<td>version</td>
<td>Specifies the version of the serialization method.</td>
<td><code>xml/xhtml: 1.0, 1.1, html: 4.0, 4.01, 5.0</code></td>
<td><code>1.0</code></td>
</tr>
<tr>
<td>html-version</td>
<td>Specifies the version of the HTML serialization method.</td>
<td><code>4.0, 4.01, 5.0</code></td>
<td><code>4.0</code></td>
</tr>
<tr>
<td>item-separator</td>
<td>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.</td>
<td><code>string</code></td>
<td></td>
</tr>
</tbody>
</table>

182
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>Encoding to be used for outputting the data.</td>
<td>all encodings supported by Java</td>
<td>UTF-8</td>
</tr>
<tr>
<td>indent</td>
<td>Adds leading whitespaces to make the output more readable.</td>
<td>yes, no</td>
<td>no Default changed with Version 10</td>
</tr>
<tr>
<td>cdata-section-elements</td>
<td>List of elements to be output as CDATA, separated by whitespaces.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>omit-xml-declaration</td>
<td>Omits the XML declaration, which is serialized before the actual query result.</td>
<td>yes, no, yes</td>
<td></td>
</tr>
<tr>
<td>standalone</td>
<td>Prints or omits the standalone attribute in the XML declaration.</td>
<td>yes, no, omit</td>
<td>omit</td>
</tr>
<tr>
<td>doctype-system</td>
<td>Introduces the output with a document type declaration and the given system identifier.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>doctype-public</td>
<td>If doctype-system is specified, adds a public identifier.</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>undeclare-prefixes</td>
<td>Undeclares prefixes in XML 1.1.</td>
<td>yes, no</td>
<td>no</td>
</tr>
<tr>
<td>normalization-form</td>
<td>Specifies a normalization form.</td>
<td>NFC, none</td>
<td>NFC</td>
</tr>
<tr>
<td>media-type</td>
<td>Specifies the media type.</td>
<td>string</td>
<td>application/xml</td>
</tr>
<tr>
<td>parameter-document</td>
<td>Parses the value as XML document with additional serialization parameters (see the Specification for more details and examples).</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>use-character-maps</td>
<td>Defines character mappings. If mappings are supplied as single string, keys and values are separated by the equal sign, and multiple pairs are separated by commas. Separators that are to be defined as keys or values can be encoded as entities. Example: A=alpha, B=beta</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
### Serialization

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte-order-mark</td>
<td>Prints a byte-order-mark before starting serialization.</td>
<td>yes, no</td>
<td>no</td>
</tr>
<tr>
<td>escape-uri-attributes</td>
<td>Escapes URI information in certain HTML attributes. Example:</td>
<td>yes, no</td>
<td>no</td>
</tr>
<tr>
<td>include-content-type</td>
<td>Inserts a meta content-type element into the head element if the result is output as HTML. Example:</td>
<td>yes, no</td>
<td>yes</td>
</tr>
</tbody>
</table>

BaseX provides some additional serialization parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>csv</td>
<td>Defines the way how data is serialized as CSV.</td>
<td>see CSV Module</td>
<td></td>
</tr>
<tr>
<td>json</td>
<td>Defines the way how data is serialized as JSON.</td>
<td>see JSON Module</td>
<td></td>
</tr>
<tr>
<td>tabulator</td>
<td>Uses tab characters (\t) instead of spaces for indenting elements.</td>
<td>yes, no</td>
<td>no</td>
</tr>
<tr>
<td>indents</td>
<td>Specifies the number of characters to be indented.</td>
<td>positive number</td>
<td>2</td>
</tr>
<tr>
<td>newline</td>
<td>Specifies the type of newline to be used as end-of-line marker.</td>
<td>\n, \r\n, \r</td>
<td>system dependent</td>
</tr>
<tr>
<td>limit</td>
<td>Stops serialization after the specified number of bytes has been serialized. If a negative number is specified, everything will be output.</td>
<td>positive number</td>
<td>-1</td>
</tr>
<tr>
<td>binary</td>
<td>Indicates if items of binary type are output in their native byte representation. Only applicable to the base serialization method.</td>
<td>yes, no</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

```xml
declare option output:method "csv";
declare option output:csv "header=yes, separator=semicolon";
<csv>
  <record>
    <Name>John</Name>
    <City>Newton</City>
  </record>
  <record>
```
If `fn:serialize` is called, output-specific parameters can be supplied via nested options:

```xml
serialize(
  <csv>
    <record>
      <Name>John</Name>
      <City>Newton</City>
    </record>
    <record>
      <Name>Jack</Name>
      <City>Oldtown</City>
    </record>
  </csv>,
  map {
    'method': 'csv',
    'csv': map { 'header': 'yes', 'separator': ';' }
  }
)
```

Result:

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

### Character mappings

Character maps allow a specific character in the instance of the data model to be replaced with a specified string of characters during serialization. The string that is substituted is output “as is,” and the serializer performs no checks that the resulting document is well-formed. This may only occur in documents parsed with `parameter-document`. If a character is mapped, then it is not subjected to XML or HTML escaping. For details, refer to section 11 Character maps in the W3C XQuery Serialization 3.1 document.

This example maps the Unicode U+00A0 NO-BREAK SPACE as `&amp;#160;` (without the serialization parameter, the Unicode character would be output):

**Example query:**

```xml
declare option output:parameter-document "map.xml";
<x>&amp;#xA0;</x>
```

**Example parameter-document:**

```xml
<serialization-parameters
  xmlns="http://www.w3.org/2010/xslt-xquery-serialization">
  <use-character-maps>
    <character-map character="&amp;#160;" map-string="&amp;amp;#160;"/>
  </use-character-maps>
</serialization-parameters>
```

### Changelog

**Version 10.0**

- Updated: `indent`: Default changed from `yes` to `no`.

**Version 9.2**

- Updated: New default value for `include-content-type` is `yes`.
Version 8.4

- Added: Serialization parameter binary.

- Updated: New serialization method basex. By default, items of binary type are now output in their native byte representation. The method raw was removed.

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, jsonunescape, 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 33. 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

- **Namespace URI:** [http://www.w3.org/2005/xqt-errors](http://www.w3.org/2005/xqt-errors)
- **Namespace prefix:** `err`
- **Codes:** `XPST`, `XQST`

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPST0003</td>
<td>An error occurred while parsing the query string (i.e., <code>1+for i in //</code> return <code>$i</code> 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.</td>
<td><code>1+for i in //* return $i</code></td>
</tr>
<tr>
<td>XPST0005</td>
<td>An expression will never return any results, no matter what input is provided.</td>
<td><code>doc('input')/..</code></td>
</tr>
<tr>
<td>XPST0008</td>
<td>A variable or type name is used that has not been defined in the current scope.</td>
<td><code>$a---element(*, x)</code></td>
</tr>
<tr>
<td>XPST0017</td>
<td>The specified function is unknown, it uses the wrong number of arguments, or, when calling Java functions: there is more than one function with the same number of arguments.</td>
<td><code>unknown()count(1,2,3)</code></td>
</tr>
<tr>
<td>XPST0051</td>
<td>An unknown QName is used in a sequence type (e.g. in the target type of the cast expression).</td>
<td><code>1 instance of x&quot;test&quot; cast as xs:itr</code></td>
</tr>
<tr>
<td>XPST0080</td>
<td><code>xs:NOTATION</code> or <code>xs:anyAtomicType</code> is used as target type of <code>cast</code> or <code>castable</code>.</td>
<td><code>1 castable as xs:NOTATION</code></td>
</tr>
<tr>
<td>XPST0081</td>
<td>A QName uses a prefix that has not been bound to any namespace, or a pragma or option declaration has not been prefixed.</td>
<td><code>unknown:x(# pragma #) { 1 }</code></td>
</tr>
<tr>
<td>XQST0009</td>
<td>The query imports a schema (schema import is not supported by BaseX).</td>
<td><code>&lt;elem xmlns=&quot;( 'dynamic' )&quot;/&gt;</code></td>
</tr>
<tr>
<td>XQST0022</td>
<td>Namespace values must be constant strings.</td>
<td><code>&lt;elem xmlns=&quot;( 'dynamic' )&quot;/&gt;</code></td>
</tr>
<tr>
<td>XQST0031</td>
<td>The specified XQuery version is not specified.</td>
<td><code>xquery version &quot;9.9&quot;;</code></td>
</tr>
<tr>
<td>XQST0032</td>
<td>The base URI was declared more than once.</td>
<td><code>declare base-uri ...</code></td>
</tr>
<tr>
<td>XQST0033</td>
<td>A namespace prefix was declared more than once.</td>
<td><code>namespace a=&quot;a&quot;;namespace a=&quot;b&quot;;</code></td>
</tr>
<tr>
<td>XQST0034</td>
<td>A function was declared more than once.</td>
<td><code>declare function local:a() { 1 };declare function local:a() { 2 }; local:a()</code></td>
</tr>
<tr>
<td>Error Code</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XQST0038</td>
<td>The default collation was declared more than once.</td>
<td>declare default collation ...</td>
</tr>
<tr>
<td>XQST0039</td>
<td>Two or more parameters in a user-defined function have the same name.</td>
<td>declare function local:fun($a, $a) { $a * $a }; local:fun(1,2)</td>
</tr>
<tr>
<td>XQDY0040</td>
<td>Two or more attributes in an element have the same node name.</td>
<td>&lt;elem a=&quot;1&quot; a=&quot;12&quot;/&gt;</td>
</tr>
<tr>
<td>XQDY0045</td>
<td>A user-defined function uses a reserved namespace.</td>
<td>declare function fn:fun() { 1 }; ()</td>
</tr>
<tr>
<td>XQST0047</td>
<td>A module was defined more than once.</td>
<td>import module ...</td>
</tr>
<tr>
<td>XQST0048</td>
<td>A module declaration does not match the namespace of the specified module.</td>
<td>import module namespace invalid=&quot;uri&quot;; 1</td>
</tr>
<tr>
<td>XQST0049</td>
<td>A global variable was declared more than once.</td>
<td>declare variable $a := 1; declare variable $a := 1; $a</td>
</tr>
<tr>
<td>XQST0054</td>
<td>A global variable depends on itself. This may be triggered by a circular variable definition.</td>
<td>declare variable $a := local:a(); declare function local:a() { $a }; $a</td>
</tr>
<tr>
<td>XQST0055</td>
<td>The mode for copying namespaces was declared more than once.</td>
<td>declare copy-namespaces ...</td>
</tr>
<tr>
<td>XQST0057</td>
<td>The namespace of a schema import may not be empty.</td>
<td>import schema &quot;&quot;; ()</td>
</tr>
<tr>
<td>XQST0059</td>
<td>The schema or module with the specified namespace cannot be found or processed.</td>
<td>import module &quot;unknown&quot;; ()</td>
</tr>
<tr>
<td>XQST0060</td>
<td>A user-defined function has no namespace.</td>
<td>declare default function namespace &quot;&quot;; declare function x() { 1 }; 1</td>
</tr>
<tr>
<td>XQST0065</td>
<td>The ordering mode was declared more than once.</td>
<td>declare ordering ...</td>
</tr>
<tr>
<td>XQST0065</td>
<td>The default namespace mode for elements or functions was declared more than once.</td>
<td>declare default element namespace ...</td>
</tr>
<tr>
<td>XQST0067</td>
<td>The construction mode was declared more than once.</td>
<td>declare construction ...</td>
</tr>
<tr>
<td>XQST0068</td>
<td>The mode for handling boundary spaces was declared more than once.</td>
<td>declare boundary-space ...</td>
</tr>
<tr>
<td>XQST0069</td>
<td>The default order for empty sequences was declared more than once.</td>
<td>declare default order empty ...</td>
</tr>
<tr>
<td>XQST0070</td>
<td>A namespace declaration overwrites a reserved namespace.</td>
<td>declare namespace xml=&quot;&quot;; ()</td>
</tr>
<tr>
<td>XQST0071</td>
<td>A namespace is declared more than once in an element constructor.</td>
<td>&lt;a xmlns=&quot;uri1&quot; xmlns=&quot;uri2&quot;/&gt;</td>
</tr>
<tr>
<td>XQST0075</td>
<td>The query contains a validate expression (validation is not supported by BaseX).</td>
<td>validate strict { () }</td>
</tr>
<tr>
<td>XQST0076</td>
<td>A group by or order by clause specifies an unknown collation.</td>
<td>for $i in 1 to 10 order by $i collation &quot;unknown&quot; return $i</td>
</tr>
<tr>
<td>XQST0079</td>
<td>A pragma was specified without the expression that is to be evaluated.</td>
<td>($ xml:a #) ()</td>
</tr>
<tr>
<td>XQST0085</td>
<td>An empty namespace URI was specified.</td>
<td>&lt;pref:elem xmlns:pref=&quot;&quot;/&gt;</td>
</tr>
<tr>
<td>XQST0087</td>
<td>An unknown encoding was specified. Note that the encoding declaration is currently ignored in BaseX.</td>
<td>xquery version &quot;1.0&quot; encoding &quot;a b&quot;; ()</td>
</tr>
<tr>
<td>XQST0088</td>
<td>An empty module namespace was specified.</td>
<td>import module &quot;&quot;; ()</td>
</tr>
<tr>
<td>XQST0089</td>
<td>Two variables in a for or let clause have the same name.</td>
<td>for $a at $a in 1 return $i</td>
</tr>
</tbody>
</table>
XQuery Errors

XQST0090 A character reference specifies an invalid character. "&#0;"

XQST0093 A module depends on itself. This may be triggered by a import module ...

circular module definition.

XQST0094 group by references a variable that has not been declared before.

for $a in l 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

• Namespace URI: http://www.w3.org/2005/xqt-errors

• Namespace prefix: err

• 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, <elem>text { attribute a as other nodes of a different type were specified before. { "val" } }</elem>

XQTY0105 A function item has been specified as content of an element.

Dynamic Errors

• Namespace URI: http://www.w3.org/2005/xqt-errors

• Namespace prefix: err

189
### XQuery Errors

- **Codes:** XPDY, XQDY

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPDY0002</td>
<td>No value has been defined for an external variable, or no context item has been set before the query was executed.</td>
<td>declare variable $x external; $xdescendant::*</td>
</tr>
</tbody>
</table>
| 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. | xs:int"string"[/]

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

### Functions Errors

- **Namespace URI:** http://www.w3.org/2005/xqt-errors

- **Namespace prefix:** err

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOAR0001</td>
<td>A value was divided by zero.</td>
<td>1 div 0</td>
</tr>
<tr>
<td>FOAR0002</td>
<td>A numeric declaration or operation causes an over- or underflow.</td>
<td>12345678901234567890xs:double(&quot;-INF&quot;) idiv 1</td>
</tr>
<tr>
<td>FOCA0002</td>
<td>A float number cannot be converted to a decimal or integer value, or a function argument cannot be converted to a valid QName.</td>
<td>xs:int(xs:double(&quot;INF&quot;)) QName(&quot;&quot;, &quot;el em&quot;)</td>
</tr>
<tr>
<td>FOCA0003</td>
<td>A value is too large to be represented as integer.</td>
<td>xs:integer(99e100)</td>
</tr>
<tr>
<td>FOCA0005</td>
<td>&quot;NaN&quot; is supplied to duration operations.</td>
<td>xs:yearMonthDuration(&quot;P1Y&quot;) * xs:double(&quot;NaN&quot;)</td>
</tr>
</tbody>
</table>
FOCH0001 A codepoint was specified that does not represent a valid XML character. codepoints-to-string(0)

FOCH0002 An unsupported collation was specified in a function. compare('a', 'a', 'unknown')

FOCH0003 An unsupported normalization form was specified in a function. normalize-unicode('a', 'unknown')

FODC0001 The argument specified in fn:id() or fn:idref() 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 format-number(1, "0", "invalid")

FODF1310 The picture string passed to fn:format-number() format-number(1, "invalid") is 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-"[]") is invalid.

FOFD1350 The picture string passed to fn:format-time(current-time(), date(), fn:format-time() or fn:format-"[Y2]") date-time() specifies a non-available component.

FONS0004 A function has a QName as argument that specifies an resolve-QName("x:e", <e/>) unbound prefix.

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 exactly-one((1, 2)) than one item.

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')
XQuery Errors

FORX0002 A function specifies an invalid regular expression.
FORX0003 A regular expression matches an empty string.
FORX0004 The replacement string of a regular expression is invalid.

FOTY0012 An item has no typed value.
FOTY0013 Functions items cannot be atomized, have no defined equality, and have no string representation.
FOTY0014 Function items have no string representation.
FOTY0015 Function items cannot be compared.

FOUT1170 Function argument cannot be used to retrieve a text resource.
FOUT1190 Encoding to retrieve a text resource is invalid or not supported.

Serialization Errors

- Namespace URI: http://www.w3.org/2005/xqt-errors
- Codes: SEPM, SERE, SESU

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESU0007</td>
<td>The specified encoding is not supported.</td>
<td>declare option output:encoding &quot;xyz&quot;; 1</td>
</tr>
<tr>
<td>SEPM0009</td>
<td>omit-xml-declaration is set to yes, and standalone has a value other than omit.</td>
<td></td>
</tr>
<tr>
<td>SEPM010</td>
<td>method is set to xml, undeclare-prefixes is set to yes, and version is set to 1.0.</td>
<td></td>
</tr>
<tr>
<td>SERE0014</td>
<td>method is set to html, and an invalid HTML character is found.</td>
<td></td>
</tr>
<tr>
<td>SERE0015</td>
<td>method is set to html, and a closing bracket (&gt;) appears inside a processing instruction.</td>
<td></td>
</tr>
<tr>
<td>SEPM0016</td>
<td>A specified parameter is unknown or has an invalid value.</td>
<td>declare option output:indent &quot;nope&quot;; 1</td>
</tr>
</tbody>
</table>

Update Errors

- Namespace URI: http://www.w3.org/2005/xqt-errors
- Codes: FOUP, XUDY, XUST, XUTY

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOUP0001</td>
<td>The first argument of fn:put() must be a document node or element.</td>
<td>fn:put(text 'file.txt') ( 1 ) ,</td>
</tr>
<tr>
<td>FOUP0002</td>
<td>The second argument of fn:put() is not a valid URI.</td>
<td>fn:put(&lt;a/&gt;, '///')</td>
</tr>
</tbody>
</table>
XUDY0009 The target node of a replace expression needs a parent in order to be replaced.

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 <nullb/>)</code>

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 copy $a := <x/> modify 1 return $a

XUST0003 The revalidation mode was declared more than once.

declare revalidation ...

XUST0004 New attributes to be inserted must directly follow the root node.

insert node (<a/>, attribute a{""}) into <a/>

XUST0005 A single element or document node is expected as target of an insert expression.

insert node <new/> into attribute a{""}
XQuery Errors

XUTY0006 A single element, text, comment or processing insert node <new/> after instruction is expected as target of an insert attribute a { "" }
before/after expression.

XUTY0007 Only nodes can be deleted.

XUTY0008 A single element, text, attribute, comment or processing replace node document ( <a/
instruction is expected as target of a replace > ) with <nullb/>></code>
expression.

XUTY0010 In a replace expression, in which no attributes are replace node <a><nullb/></a>/
targeted, the replacing nodes must not be attributes as b with attribute size { 1 }/</code>

XUTY0011 In the replace expression, in which attributes are replace node <e a=""/>/@a targeted, the replacing nodes must be attributes as well. with <a/>

XUTY0012 In a rename expression, the target nodes must be an rename node text { 1 } as <x/>

XUTY0013 An expression in the copy clause must return a single node.

XUTY0022 An attribute must not be inserted into a document node.

Full-Text Errors

• Namespace URI: http://www.w3.org/2005/xqt-errors
• Namespace prefix: err
• Codes: FTDY, FTST

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

BaseX Errors

• Namespace URI: http://basex.org
• Namespace prefix: basex

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Annotation</td>
<td>Annotation errors.</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td></td>
</tr>
<tr>
<td>doc</td>
<td>The argument specified via <code>fn:doc</code> must yield a single document.</td>
<td></td>
</tr>
<tr>
<td>error</td>
<td>Generic error, which is e.g. raised by Java bindings.</td>
<td></td>
</tr>
<tr>
<td>function</td>
<td>Function items cannot be cached.</td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>The function was called outside an HTTP servlet context.</td>
<td></td>
</tr>
<tr>
<td>options</td>
<td>The specified database option is unknown.</td>
<td></td>
</tr>
<tr>
<td>overflow</td>
<td>Stack overflow.</td>
<td></td>
</tr>
<tr>
<td>permission</td>
<td>The current user has insufficient permissions to open a database, update nodes, etc.</td>
<td></td>
</tr>
<tr>
<td>restxq</td>
<td>Errors related to RESTXQ.</td>
<td></td>
</tr>
<tr>
<td>update</td>
<td>BaseX-specific update errors.</td>
<td></td>
</tr>
</tbody>
</table>

Additional, module-specific error codes are listed in the descriptions of the query modules.
Part VII. XQuery Modules
Chapter 34. 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 and errors in this module are assigned to the http://basex.org/modules/admin namespace, which is statically bound to the admin prefix.

Database Logs

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($text as xs:string) as empty-sequence(), admin:write-log($text as xs:string, $type as xs:string) as empty-sequence() |
| Summary | Writes a string to the database logs, along with current user data (timestamp, username). An optional log $type can be specified. If omitted, the log type is INFO. If the function is called from a database client, the IP will be logged. Otherwise, the string SERVER will be logged. |
| Errors | type: Type string contains whitespaces. |

admin:delete-logs

| Signatures | admin:delete-logs($date as xs:string) as empty-sequence() |
| Summary | Deletes the log entries from the specified $date |
| Errors | today: Today's log file cannot be deleted.delete: An error occurred while deleting a log file. |

Database Sessions

admin:sessions

| Signatures | admin:sessions() as element(session)* |
Admin Module

Summary

Returns an element sequence with all currently opened database sessions, including the username, 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"/>

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>An error occurred while deleting a log file.</td>
</tr>
<tr>
<td>today</td>
<td>Today's log file cannot be deleted.</td>
</tr>
<tr>
<td>type</td>
<td>Type string contains whitespaces.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.2

- Updated: `admin:write-log`: type string may contain more characters

Version 9.0

- Updated: error codes updated; errors now use the module namespace

Version 8.3

- Updated: `admin:write-log`: optional log type added

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

Content Handling

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

- `error`: archive creation failed.

Examples

Sums up the file sizes of all entries of a JAR file:

```
sum(archive:entries(file:read-binary('jar.jar'))/@size)
```

archive:options

Signatures

```
archive:options($archive as xs:base64Binary) as map(*).
```

Summary

Returns the options of the specified `$archive` in the format specified by `archive:create`.

Errors

- `format`: The archive format is not supported.
- `error`: archive creation failed.

Examples

A standard ZIP archive will return the following options:

```
map {
  "format": "zip",
  "algorithm": "deflate"
}
```

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, $options as map(*)) as xs:string*,
archive:extract-text($archive as xs:base64Binary, $options as map(*), $callback as function()) as xs:string*.
```

Summary

Expects an archive and a list of entry descriptors to extract. If the `$options` parameter is specified, the entries will be extracted according to the given options. The `$callback` parameter is a function that is called for each entry that is extracted.

Errors

- `error`: archive extraction failed.

Examples

Extracts all text from an archive:

```
archive:extract-text(file:read-binary('archive.zip'))
```

Extracts the text of a specific entry:

```
archive:extract-text(file:read-binary('archive.zip'), doc('index.html'))
```

If a function is passed as `$callback` parameter, the function is called for each entry that is extracted. The function will receive an entry as an argument:

```
defile:callback(entry as element(archive:entry)) {
  printf
  ...}
```
archive:extract-text

Signatures
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
e: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if CHECKSTRINGS is turned off. err: archive creation failed.

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($archive as xs:base64Binary) as xs:base64Binary*, archive:extract-binary($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
er: archive creation failed.

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)
return for-each-pair($entries, $contents, function($entry, $content) {
  file:create-dir(replace($entry, "[^/]+$", "")),
  file:write-binary($entry, $content)
})

Updates
archive:create

Signatures
archive:create($entries as item(), $contents as item()* as xs:base64Binary, $options as map(*)?) 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:
• `format`: allowed values are `zip` and `gzip`. `zip` is the default.

• `algorithm`: allowed values are `deflate` and `stored` (for the `zip` format). `deflate` is the default.

**Errors**

- `number`: the number of entries and contents differs.
- `format`: the specified option or its value is invalid or not supported.
- `descriptor`: entry descriptors contain invalid entry names, timestamps or compression levels.
- `encode`: the specified encoding is invalid or not supported, or the string conversion failed.
- `single`: the chosen archive format only allows single entries.
- `error`: archive creation failed.

**Examples**

The following one-liner creates an archive `archive.zip` with one file `file.txt`:

```xml
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:

```xml
let $path  := 'audio/'
let $files := file:list($path, true(), '*.mp3')
let $zip   := archive:create($files,
  for $file in $files
    return file:read-binary($path || $file)
)
return file:write-binary('mp3.zip', $zip)
```

### archive:delete

**Signatures**

```xml
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**

- `modify`: the entries of the given archive cannot be modified.
- `error`: archive creation failed.

**Examples**

This example deletes all HTML files in an archive and creates a new file:

```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)
```
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))

### Convenience

#### archive:create-from

**Signatures**

archive:create-from($path as xs:string) as xs:base64Binary,
archive:create-from($path as xs:string, $options as map(*)?) as xs:base64Binary,
archive:create-from($path as xs:string, $options as map(*)?, $entries as item()) as xs:base64Binary

**Summary**

This convenience function creates an archive from all files in the specified directory $path. The $options parameter contains archiving options, and the files to be archived can be limited via $entries. The format of the two last arguments is identical to archive:create, but two additional options are available:

- recursive: parse all files recursively (default: true; ignored if entries are specified via the last argument).
- root-dir: use name of supplied directory as archive root directory (default: false).

**Errors**

file:no-dir: the specified path does not point to a directory.
file:is-dir: one of the specified entries points to a directory.
file:not-found: a specified entry does not exist.
error: archive creation failed.

**Examples**

This example writes the files of a user’s home directory to archive.zip:

let $zip := archive:create-from('/home/user/)
return file:write-binary('archive.zip', $zip)

#### archive:extract-to

**Signatures**

archive:extract-to($path as xs:string, $archive as xs:base64Binary) as empty-sequence(),
archive:extract-to($path as xs:string, $archive as xs:base64Binary, $entries as item()) as empty-sequence()

**Summary**

This convenience function writes files of an $archive directly to the specified directory $path. The archive entries to be written can be restricted via $entries. The format of the argument is the same as for archive:create (attributes will be ignored).

**Errors**

error: archive creation failed.

**Examples**

The following expression unzips all files of an archive to the current directory:

archive:extract-to('.', file:read-binary('archive.zip'))

#### archive:write

**Signatures**

archive:write($path as xs:string, $entries as item()), $contents as item()) as xs:base64Binary,
archive:write($path as xs:string, $entries as item()), $contents as item()*, $options as map(*)?) as xs:base64Binary.

**Summary**

This convenience function creates a new archive from the specified $entries and $contents and writes it disk. See archive:create for more details.

**Errors**

number: the number of entries and contents differs.
format: the specified option or its value is invalid or not supported.
descriptor: entry descriptors contain invalid entry names, timestamps or compression levels.
encode: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if CHECKSTRINGS is turned off.
single: the chosen archive format only allows single entries.
error: archive creation failed.
Examples

All mp3 files from a directory are zipped and written to a file, along with an info file:

```xml
let $files := file:children('music')[ends-with(., 'mp3')] return archive:write(
    'music.zip',
    ('info.txt', $files ! file:name(.)),
    ('Archive with MP3 files', $files ! file:read-binary(.))
)
```

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>descriptor</td>
<td>Entry descriptors contain invalid entry names, timestamps or compression levels.</td>
</tr>
<tr>
<td>encode</td>
<td>The specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if CHECKSTRINGS is turned off.</td>
</tr>
<tr>
<td>error</td>
<td>processing failed.</td>
</tr>
<tr>
<td>format</td>
<td>The archive format or the specified option is invalid or not supported.</td>
</tr>
<tr>
<td>modify</td>
<td>The entries of the given archive cannot be modified.</td>
</tr>
<tr>
<td>number</td>
<td>The number of specified entries and contents differs.</td>
</tr>
<tr>
<td>single</td>
<td>The chosen archive format only allows single entries.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.6

• Added: `archive:write`

Version 9.0

• Updated: `archive:create-from`: options added
• Updated: error codes updated; errors now use the module namespace

Version 8.5

• Updated: `archive:options`: map returned instead of element

Version 8.3

• Added: `archive:create-from, archive:extract-to (replaces archive:write)`

The module was introduced with Version 7.3.
Chapter 36. Array Module

This XQuery Module contains functions for manipulating arrays, which has been introduced with XQuery 3.1.

Conventions

All functions and errors 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(array { 1 to 10 }) returns 10.
- array:size([1 to 10]) returns 1, because the array contains a single sequence with 10 integers.

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:subarray(["a", "b", "c"], 2) returns the array ["b", "c"].

array:put

**Signatures**

array:put($array as array(*), $position as xs:integer, $member as item()*) as array(*)
### Array Module

<table>
<thead>
<tr>
<th>Function</th>
<th>Signatures</th>
<th>Summary</th>
<th>Errors</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>array:put</strong></td>
<td>array:put($array as array(<em>), $position as xs:integer, $member as item()) as array(</em>)</td>
<td>Returns a copy of $array with $member replaced at the specified $position. Equivalent to $array =&gt; array:remove($position) =&gt; array:insert-before($position, $member).</td>
<td>FOAY0001: $position is not in the range 1 to array:size($array) + 1 inclusive.</td>
<td>• array:put([&quot;a&quot;, &quot;b&quot;, &quot;c&quot;], 2, &quot;d&quot;) returns the array [&quot;a&quot;, &quot;d&quot;, &quot;c&quot;].</td>
</tr>
<tr>
<td><strong>array:remove</strong></td>
<td>array:remove($array as array(<em>), $positions as xs:integer</em>) as array(*)</td>
<td>Returns a copy of $array without the member at the specified $positions.</td>
<td>FOAY0001: A position is not in the range 1 to array:size($array) + 1 inclusive.</td>
<td>• array:append([&quot;a&quot;], 1) returns the array [].</td>
</tr>
<tr>
<td><strong>array:insert-before</strong></td>
<td>array:insert-before($array as array(<em>), $position as xs:integer, $member as item()) as array(</em>)</td>
<td>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).</td>
<td>FOAY0001: $position is not in the range 1 to array:size($array) + 1 inclusive.</td>
<td>• array:insert-before([&quot;a&quot;], 1, &quot;b&quot;) returns the array [&quot;b&quot;, &quot;a&quot;].</td>
</tr>
<tr>
<td><strong>array:head</strong></td>
<td>array:head($array as array(<em>)) as item(</em>)</td>
<td>Returns the first member of $array. This function is equivalent to the expression $array(1).</td>
<td>FOAY0001: The array is empty.</td>
<td>• array:head([&quot;a&quot;, &quot;b&quot;]) returns &quot;a&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• array:head([&quot;a&quot;, &quot;b&quot;, &quot;c&quot;, &quot;d&quot;]) returns the array [&quot;a&quot;, &quot;b&quot;].</td>
</tr>
<tr>
<td><strong>array:tail</strong></td>
<td>array:tail($array as array(<em>)) as array(</em>)</td>
<td>Returns a new array with all members except the first from $array. This function is equivalent to the expression array:remove($array, 1).</td>
<td>FOAY0001: The array is empty.</td>
<td>• array:insert-before([&quot;a&quot;], 1, &quot;b&quot;) returns the array [&quot;b&quot;, &quot;a&quot;].</td>
</tr>
<tr>
<td><strong>array:reverse</strong></td>
<td>array:reverse($array as array(<em>)) as array(</em>)</td>
<td>Returns a new array with all members of $array in reverse order.</td>
<td></td>
<td>• array:reverse(array { 1 to 3 }) returns the array [3, 2, 1].</td>
</tr>
<tr>
<td><strong>array:join</strong></td>
<td>array:join($arrays as array(<em>)) as array(</em>)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

205
| **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 $array. |
| **Examples** | The following query returns the array `[2, 3, 4, 5, 6]`:  
  ```java  
  array:for-each(  
    array { 1 to 5 },  
    function($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 $function returns true. |
| **Examples** | The following query returns the array `[0, 1, 3]`:  
  ```java  
  array:filter(  
    array { 0, 1, -2, 3, -4 },  
    function($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 $zero as first argument. |
| **Examples** | The following query returns 55 (the sum of the integers 1 to 10):  
  ```java  
  array:fold-left(  
    array { 1 to 10 },  
    0,  
    function($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()*` |
Array Module

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 }):

```xml
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]:

```xml
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(*), $collation as xs:string?) as array(*), array:sort($array as array(*), $collation as xs:string?, $key as function(item()* as xs:anyAtomicType*) as array(*))

Summary
Returns a new array with sorted $array members, using an optional $collation. If a $key function is supplied, 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)

Errors

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

Changelog

Version 8.6
• Updated: array:put collation argument was inserted between first and second argument.

Version 8.5
• Added: array:put

Version 8.4

• Removed: array:serialize (use fn:serialize instead)

Introduced with Version 8.0.
Chapter 37. 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

<table>
<thead>
<tr>
<th>Signatures</th>
<th>bin:hex($in as xs:string?) as xs:base64Binary?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>non-numeric-character: the input cannot be parsed as a hexadecimal number.</td>
</tr>
<tr>
<td>Examples</td>
<td>string(bin:hex('11223F4E')) yields ESI/Tg==. string(xs:hexBinary(bin:hex('FF'))) yields FF.</td>
</tr>
</tbody>
</table>

### bin:bin

<table>
<thead>
<tr>
<th>Signatures</th>
<th>bin:bin($in as xs:string?) as xs:base64Binary?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>non-numeric-character: the input cannot be parsed as a binary number.</td>
</tr>
<tr>
<td>Examples</td>
<td>string(bin:bin('1101000111010101')) yields OdU=. string(xs:hexBinary(bin:bin('100111010101'))) yields 11D5.</td>
</tr>
</tbody>
</table>

### bin:octal

<table>
<thead>
<tr>
<th>Signatures</th>
<th>bin:octal($in as xs:string?) as xs:base64Binary?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>non-numeric-character: the input cannot be parsed as an octal number.</td>
</tr>
<tr>
<td>Examples</td>
<td>string(xs:hexBinary(bin:octal('11223047'))) yields 252627.</td>
</tr>
</tbody>
</table>
### Binary Module

**bin:to-octets**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>bin:to-octets($in as xs:base64Binary) as xs:integer*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

**bin:from-octets**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>bin:from-octets($in as xs:integer*) as xs:base64Binary</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>octet-out-of-range: one of the octets lies outside the range 0 - 255.</td>
</tr>
</tbody>
</table>

### Basic Operations

**bin:length**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>bin:length($in as xs:base64Binary) as xs:integer</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the size of binary data in octets.</td>
</tr>
</tbody>
</table>

**bin:part**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>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?</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>negative-size: the specified size is negative. index-out-of-range: the specified offset + size is out of range.</td>
</tr>
<tr>
<td>Examples</td>
<td>Test whether binary data starts with binary content consistent with a PDF file: <code>bin:part($data, 0, 4) eq bin:hex(&quot;25504446&quot;)</code>.</td>
</tr>
</tbody>
</table>

**bin:join**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>bin:join($in as xs:base64Binary*) as xs:base64Binary</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns an <code>xs:base64Binary</code> 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.</td>
</tr>
</tbody>
</table>

**bin:insert-before**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>bin:insert-before($in as xs:base64Binary?, $offset as xs:integer, $extra as xs:base64Binary?) as xs:base64Binary?</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>index-out-of-range: the specified offset is out of range.</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>$in as xs:double</th>
<th>as xs:base64Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$in as xs:double, $octet-order as xs:string</td>
<td>as xs:base64Binary</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>$in as xs:float</th>
<th>as xs:base64Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$in as xs:float, $octet-order as xs:string</td>
<td>as xs:base64Binary</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>$in as xs:integer, $size as xs:integer</th>
<th>as xs:base64Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>$in as xs:integer, $size as xs:string, $octet-order as xs:string</td>
<td>as xs:base64Binary</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>$in as xs:base64Binary, $offset as xs:integer</th>
<th>as xs:double</th>
</tr>
</thead>
<tbody>
<tr>
<td>$in as xs:base64Binary, $offset as xs:string, $octet-order as xs:string</td>
<td>as xs:double</td>
</tr>
</tbody>
</table>

**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.
## Binary Module

### 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?
```
Binary Module

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>differing-length-arguments</td>
<td>The arguments to a bitwise operation have different lengths.</td>
</tr>
<tr>
<td>index-out-of-range</td>
<td>An offset value is out of range.</td>
</tr>
<tr>
<td>negative-size</td>
<td>A size value is negative.</td>
</tr>
<tr>
<td>octet-out-of-range</td>
<td>An octet value lies outside the range 0-255.</td>
</tr>
<tr>
<td>non-numeric-character</td>
<td>Binary data cannot be parsed as number.</td>
</tr>
<tr>
<td>unknown-encoding</td>
<td>An encoding is not supported.</td>
</tr>
<tr>
<td>conversion-error</td>
<td>An error or malformed input during converting a string.</td>
</tr>
<tr>
<td>unknown-significance-order</td>
<td>An octet-order value is unknown.</td>
</tr>
</tbody>
</table>

Changelog

 Introduced with Version 7.8.
Chapter 38. Client Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to access BaseX server instances from XQuery. With this module, you can execute database commands and evaluate XQuery expressions.

Please note that the client module should always be used to address independent BaseX server instances. You can create deadlocks if you evaluate a query with a server instance, and if you are addressing the same server instance in your query. See the following example:

| (: Retrieve documents from database :) |
| let $client-id := client:connect('localhost', 1984, 'admin', '...') |
| let $docs := client:query($client-id, 'db:get("conflict")') |
| (: Create database with same name :) |
| return db:create('conflict', $docs, $docs ! db:path(.)) |

The read-only query cannot be processed, because the conflict database is currently write-locked by the main query. See Transaction Management for more background information.

Conventions

All functions and errors in this module are assigned to the http://basex.org/modules/client namespace, which is statically bound to the client 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 | connect: 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 a 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 | error: an I/O error occurs while transferring data from or to the server. command: 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', '...') ! |
| | client:execute(., 'create database TEST') |

client:info

| Signatures | client:info($id as xs:anyURI) as xs:string |
This function returns an information string, created by the last call of `client:execute`. `$id` specifies the session id.

### client:query

#### Signatures

```xml
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

- `error`: an I/O error occurs while transferring data from or to the server.
- `query`: an error occurs while evaluating a query, and if the original error cannot be extracted from the returned error string.
- `function`: function items (including maps and arrays) cannot be returned.

#### Examples

The following query sends a query on a local server instance, binds the integer 123 to the variable `$n` and returns 246:

```xml
let $c := client:connect('localhost', 1984, 'admin', '...')
return client:query($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:

```xml
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

```xml
client:close($id as xs:anyURI) as empty-sequence()
```

#### Summary

This function closes a client session. `$id` specifies the session id. Opened connections will automatically be closed after the XQuery expression has been evaluated, but it is recommendable to explicitly close them with this function if you open many connections.

#### Errors

- `error`: an I/O error occurs while transferring data from or to the server.

### Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>command</code></td>
<td>An error occurred while executing a command.</td>
</tr>
<tr>
<td><code>connect</code></td>
<td>An error occurred while creating a new session (possible reasons: server not available, access denied).</td>
</tr>
<tr>
<td><code>error</code></td>
<td>An I/O error occurred while transferring data from or to the server.</td>
</tr>
<tr>
<td><code>function</code></td>
<td>Function items (including maps and arrays) cannot be returned.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>The id with the specified session is unknown, or has already been closed.</td>
</tr>
<tr>
<td><code>query</code></td>
<td>An error occurred while evaluating a query. Will only be raised if the XQuery error cannot be extracted from the returned error string.</td>
</tr>
</tbody>
</table>
Changelog

Version 9.0
• Updated: error codes updated; errors now use the module namespace

Version 8.0
• Updated: Bound values may now contain no or more than one item in \texttt{client:query}.

Version 7.5
• Added: \texttt{client:info}

The module was introduced with Version 7.3.
Chapter 39. Conversion Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to convert data between different formats.

Conventions

All functions and errors in this module are assigned to the http://basex.org/modules/convert namespace, which is statically bound to the convert 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, $encoding as xs:string) as xs:string,
convert:binary-to-string($bytes as xs:anyAtomicType, $encoding as xs:string, $fallback as xs:boolean) as xs:string

Summary

Converts the specified $bytes (xs:base64Binary, xs:hexBinary) to a string:

• The UTF-8 default encoding can be overwritten with the optional $encoding argument.
• By default, invalid characters will be rejected. If $fallback is set to true, these characters will be replaced with the Unicode replacement character FFFD (#).

Errors

string: The input is an invalid XML string, or the wrong encoding has been specified.BXCO0002: The specified encoding is invalid or not supported.

Examples

• convert:binary-to-string(xs:hexBinary('48656c6c6f576f726c64')) yields HelloWorld.

convert:string-to-base64

Signatures

convert:string-to-base64($string as xs:string) as xs:base64Binary,
convert:string-to-base64($string as xs:string, $encoding as xs:string) as xs:base64Binary

Summary

Converts the specified $string to an xs:base64Binary item. If the default encoding is chosen, conversion will be cheap, as strings and binaries are both internally represented as byte arrays. The UTF-8 default encoding can be overwritten with the optional $encoding argument.

Errors

binary: The input cannot be represented in the specified encoding.encoding: The specified encoding is invalid or not supported.

Examples

• string(convert:string-to-base64('HelloWorld')) yields SGVsbG9Xb3JsZA==.

convert:string-to-hex

Signatures

convert:string-to-hex($string as xs:string) as xs:hexBinary,
convert:string-to-hex($string as xs:string, $encoding as xs:string) as xs:hexBinary

Summary

Converts the specified $string to an xs:hexBinary item. If the default encoding is chosen, conversion will be cheap, as strings and binaries are both internally represented as byte arrays. The UTF-8 default encoding can be overwritten with the optional $encoding argument.

Errors

binary: The input cannot be represented in the specified encoding.encoding: The specified encoding is invalid or not supported.
## Conversion Module

| Examples | • `string(convert:string-to-hex('HelloWorld'))` yields 48656C666F726C64. |

### Binary Data

#### convert:integers-to-base64

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>convert:integers-to-base64($integers as xs:integer*)</code> as <code>xs:base64Binary</code></th>
</tr>
</thead>
</table>
| Summary | Converts the specified $integers to an item of type `xs:base64Binary`:  
  • Only the first 8 bits of the supplied integers will be considered.  
  • Conversion of byte sequences is very efficient, as items of binary type are internally represented as byte arrays. |
| Examples | • `convert:integers-to-base64(Q{java:java.lang.String}get-bytes('abc'))` converts a byte sequence to a `xs:base64Binary` item. |

#### convert:integers-to-hex

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>convert:integers-to-hex($integers as xs:integer*)</code> as <code>xs:hexBinary</code></th>
</tr>
</thead>
</table>
| Summary | Converts the specified $integers to an item of type `xs:hexBinary`:  
  • Only the first 8 bits of the supplied integers will be considered.  
  • Conversion of byte sequences is very efficient, as items of binary type are internally represented as byte arrays. |

#### convert:binary-to-integers

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>convert:binary-to-integers($binary as xs:anyAtomicType)</code> as <code>xs:integer*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the specified $binary (xs:base64Binary, xs:hexBinary) as a sequence of unsigned integers (octets).</td>
</tr>
<tr>
<td>Examples</td>
<td>• <code>convert:binary-to-integers(xs:hexBinary('FF'))</code> yields 255.</td>
</tr>
</tbody>
</table>

#### convert:binary-to-bytes

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>convert:binary-to-bytes($binary as xs:anyAtomicType)</code> as <code>xs:byte*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the specified $binary (xs:base64Binary, xs:hexBinary) as a sequence of bytes. The conversion is very cheap and takes no additional memory, as items of binary type are internally represented as byte arrays.</td>
</tr>
</tbody>
</table>
| Examples | • `convert:binary-to-bytes(xs:base64Binary('QmFzZVggXMgY29vbA=='))` yields the sequence (66, 97, 115, 101, 88, 32, 105, 115, 32, 99, 111, 111, 108).  
  • `convert:binary-to-bytes(xs:hexBinary("4261736558"))` yields the sequence (66 97 115 101 88). |

### Numbers

#### convert:integer-to-base

| Signatures | `convert:integer-to-base($number as xs:integer, $base as xs:integer)` as `xs:string`. |
Conversion Module

Summary
Converts $number to a string, using the specified $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.

Errors
base: The specified base is not in the range 2-36.

Examples
- convert:integer-to-base(-1, 16) yields 'ffffffffffffffff'.
- convert:integer-to-base(22, 5) yields '42'.

convert:integer-from-base

Signatures
convert:integer-from-base($string as xs:string, $base as xs:integer) as xs:integer

Summary
Decodes an integer from $string, using the specified $base. The first base elements of the sequence '0',..,'9','a',..,'z' are allowed as digits; case does not matter. Valid bases are 2 - 36. If the supplied string contains more than 64 bits of information, the result will be truncated.

Errors
base: The specified base is not in the range 2-36.integer: The specified digit is not valid for the given range.

Examples
- convert:integer-from-base('CAFEBABE', 16) yields 3405691582.
- convert:integer-from-base('42', 5) yields 22.
- convert:integer-from-base(convert:integer-to-base(123, 7), 7) yields 123.

Dates and Durations

convert:integer-to-dateTime

Signatures
convert:integer-to-dateTime($milliseconds 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) yields 1970-01-01T00:00:00Z.
- convert:integer-to-dateTime(prof:current-ms()) returns the current miliseconds in the xs:dateTime format.

convert:dateTime-to-integer

Signatures
convert:dateTime-to-integer($dateTime as xs:dateTime) as xs:integer

Summary
Converts the specified $dateTime item to the number of milliseconds since 1 Jan 1970.

Examples
- convert:dateTime-to-integer(xs:dateTime('1970-01-01T00:00:00Z')) yields 0.

convert:integer-to-dayTime

Signatures
convert:integer-to-dayTime($milliseconds as xs:integer) as xs:dayTimeDuration

Examples
Conversion Module

**Summary**
Converts the specified number of $milliseconds to an item of type xs:dayTimeDuration.

**Examples**
- `convert:integer-to-dayTime(1234)` yields PT1.234S.

**convert:dayTime-to-integer**

**Signatures**
`convert:dayTime-to-integer($dayTime as xs:dayTimeDuration) as xs:integer`

**Summary**
Converts the specified $dayTime duration to milliseconds represented by an integer.

**Examples**
- `convert:dayTime-to-integer(xs:dayTimeDuration('PT1S'))` yields 1000.

**Keys**

**convert:encode-key**

**Signatures**
`convert:encode-key($key as xs:string) as xs:string, convert:encode-key($key as xs:string, $lax as xs:boolean) as xs:string`

**Summary**
Encodes the specified $key (with the optional $lax conversion method) to a valid NCName representation, which can be used to create an element node:

- An empty string is converted to a single underscore (_).
- Existing underscores are rewritten to two underscores (__).
- Characters that are no valid NCName characters are rewritten to an underscore and the character’s four-digit Unicode. For example, the exclamation mark ? is transformed to _003f.
- If lax conversion is chosen, invalid characters are replaced with underscores or (when invalid as first character of an element name) prefixed with an underscore. The resulting string may be better readable, but it cannot necessarily be converted back to the original form.

This encoding is employed by the direct conversion format in the JSON Module and the CSV Module.

**Examples**
- `element { convert:encode-key("!") } { } creates a new element with an encoded name: &lt;_0021/>.`

**convert:decode-key**

**Signatures**
`convert:decode-key($key as xs:string) as xs:string, convert:decode-key($key as xs:string, $lax as xs:boolean) as xs:string`

**Summary**
Decodes the specified $key (with the optional $lax conversion method) to the original string representation. Keys supplied to this function are usually element names from documents that have been created with the JSON Module or CSV Module.

**Examples**
- `convert:decode-key(name(&lt;_0021/>))` yields !.
- `json:doc("doc.json")/*/! convert:decode-key(name()) yields the original string representation of all names of a JSON document.`

**Errors**
- `key`: The specified key cannot be decoded to its original representation.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>The specified base is not in the range 2-36.</td>
</tr>
<tr>
<td>binary</td>
<td>The input cannot be converted to a binary representation.</td>
</tr>
</tbody>
</table>
### Changelog

**Version 9.4**
- Added: `convert:encode-key, convert:decode-key`.

**Version 9.0**
- Added: `convert:binary-to-integers`.
- Updated: `convert:bytes-to-base64, convert:integers-to-base64`: Renamed from `convert:byte-to-base64`; argument type relaxed from `xs:byte` to `xs:integer`.
- Updated: error codes updated; errors now use the module namespace

**Version 8.5**
- Updated: `convert:binary-to-string`: `$fallback` argument added.

**Version 7.5**

The module was introduced with Version 7.3. Some of the functions have been adopted from the obsolete Utility Module.
Chapter 40. 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($data as xs:anyAtomicType, $key as xs:anyAtomicType, $algorithm as xs:string) as xs:string, crypto:hmac($data as xs:anyAtomicType, $key as xs:anyAtomicType, $algorithm as xs:string, $encoding as xs:string) as xs:string |
| Summary | Creates an authentication code for the specified $data via a cryptographic hash function: |
| | • $key must not be empty. |
| | • $algorithm describes the hash algorithm which is used for encryption. Currently supported are md5, sha1, sha256, sha384, sha512. Default is md5. |
| | • $encoding must either be hex or base64; it specifies the encoding of the returned authentication code. Default is base64. |
| 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 | Return message authentication code (MAC) for a given string: Query: crypto:hmac('message', 'secretkey', 'md5', 'hex') Result: 34D1E3818B347252A75A4F6D747B21C2 |

Encryption & Decryption

The encryption and decryption functions underlie several limitations:

• Cryptographic algorithms are currently limited to symmetric algorithms. 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($data as xs:anyAtomicType, $type as xs:string, $key as xs:anyAtomicType, $algorithm as xs:string) as xs:base64Binary

**Summary**

Encrypts data with the specified key:

- $data must be a string or binary item.
- $type must be symmetric.
- $key is the secret key which is used for both encryption and decryption of input data. It must be a string or binary item. 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. 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**

Encrypt input data:

crypto:encrypt('message', 'symmetric', 'keykeyke', 'DES')

**Crypto:decrypt**

**Signatures**

crypto:decrypt($data as xs:anyAtomicType, $type as xs:string, $key as xs:anyAtomicType, $algorithm as xs:string) as xs:string

**Summary**

Encrypts data with the specified key:

- $data must be a string or binary item.
- $type must be symmetric.
- $key is the secret key which is used for both encryption and decryption of input data. It must be a string or binary item. 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. 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**

Decrypt input data and return 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** 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:

```xml
<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() |

225
# Cryptographic Module

## Summary

$\texttt{canonicalization}$ must either be $\texttt{inclusive-with-comments}$, $\texttt{inclusive}$, $\texttt{exclusive-with-comments}$ or $\texttt{exclusive}$. Default is $\texttt{inclusive-with-comments}$. $\texttt{digest}$ must be one of the following: SHA1, SHA256 or SHA512. Default is $\texttt{SHA1}$. $\texttt{signature}$ must either be $\texttt{RSA_SHA1}$ or $\texttt{DSA_SHA1}$. Default is $\texttt{RSA_SHA1}$. $\texttt{prefix}$ may be empty and prefixes the $\texttt{Signature}$ element accordingly. $\texttt{type}$ is the signature type. It must either be enveloped or enveloping (detached signatures are not supported so far). Default is $\texttt{enveloped}$. $\texttt{xpath}$ is an arbitrary XPath expression which specifies a subset of the document that is to be signed. $\texttt{certificate}$ is the digital certificate used to sign the input document. $\texttt{ext}$ may either be an $\texttt{xpath}$ expression or a $\texttt{certificate}$.

## Errors

- **CX0001**: the canonicalization algorithm is not supported.
- **CX0002**: the digest algorithm is not supported.
- **CX0003**: the signature algorithm is not supported.
- **CX0004**: the $\texttt{xpath}$ expression is invalid.
- **CX0005**: the root name of $\texttt{digital-certificate}$ 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.

## Example

Generate **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>9hvH4qztn1YgYfJDRlnEMPJoaoY==</DigestValue>
    </Reference>
    <SignatureValue>Pn/Jr44WBcdARff2UVYEiWyW1563XdqnU87nusAIaHgzd+U3SrjVJhPFLDeUDfjXvYzI5FaznTYE
P3ddeoFmyA==</SignatureValue>
    <KeyInfo>
      <RSAKeyValue>
        <Modulus>rtvpFSbCIE2BJePlVYLIRIjX10R7ESr2+D+J0VKn77M7V2bcbDRPeqRbjSkEz1HWC/N067tjB3gh4/4PPT9bgQ==</Modulus>
        <Exponent>AQAB</Exponent>
      </RSAKeyValue>
    </KeyInfo>
  </SignedInfo>
</Signature>
</a>
```

**crypto:validate-signature**

## Signatures

```
crypto:validate-signature($input-doc as node()) as xs:boolean
```

## Summary

Checks if the given node contains a $\texttt{Signature}$ element and whether the signature is valid. In this case true is returned. If the signature is invalid the function returns false.
**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX0015</td>
<td>the signature element cannot be found.</td>
</tr>
<tr>
<td>CX9994</td>
<td>an unspecified problem occurs during validation.</td>
</tr>
<tr>
<td>CX9996</td>
<td>an IO exception occurs during validation.</td>
</tr>
</tbody>
</table>

**Example**

Validate XML Signature: Query:

```xml
let $sig := crypto:generate-signature(<a/>, '', '', '', '', '')
return crypto:validate-signature($sig)
```

**Result:**

true

---

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CX0001</td>
<td>The canonicalization algorithm is not supported.</td>
</tr>
<tr>
<td>CX0002</td>
<td>The digest algorithm is not supported.</td>
</tr>
<tr>
<td>CX0003</td>
<td>The signature algorithm is not supported.</td>
</tr>
<tr>
<td>CX0004</td>
<td>The XPath expression is invalid.</td>
</tr>
<tr>
<td>CX0005</td>
<td>The root element of argument $digital-certificate must have the name 'digital-certificate'.</td>
</tr>
<tr>
<td>CX0006</td>
<td>The child element of argument $digital-certificate having position $position must have the name $child-element-name.</td>
</tr>
<tr>
<td>CX0007</td>
<td>The keystore is null.</td>
</tr>
<tr>
<td>CX0008</td>
<td>I/O error while reading keystore.</td>
</tr>
<tr>
<td>CX0009</td>
<td>Permission denied to read keystore.</td>
</tr>
<tr>
<td>CX0010</td>
<td>The keystore URL is invalid.</td>
</tr>
<tr>
<td>CX0011</td>
<td>The keystore type is not supported.</td>
</tr>
<tr>
<td>CX0012</td>
<td>Cannot find key for alias in given keystore.</td>
</tr>
<tr>
<td>CX0013</td>
<td>The hashing algorithm is not supported.</td>
</tr>
<tr>
<td>CX0014</td>
<td>The encoding method is not supported.</td>
</tr>
<tr>
<td>CX0015</td>
<td>Cannot find Signature element.</td>
</tr>
<tr>
<td>CX0016</td>
<td>No such padding.</td>
</tr>
<tr>
<td>CX0017</td>
<td>Incorrect padding.</td>
</tr>
<tr>
<td>CX0018</td>
<td>The encryption type is not supported.</td>
</tr>
<tr>
<td>CX0019</td>
<td>The secret key is invalid.</td>
</tr>
<tr>
<td>CX0020</td>
<td>Illegal block size.</td>
</tr>
<tr>
<td>CX0021</td>
<td>The algorithm is not supported.</td>
</tr>
<tr>
<td>CX0022</td>
<td>An invalid certificate alias is specified. Added to the official specification.</td>
</tr>
<tr>
<td>CX0024</td>
<td>The algorithm is invalid.</td>
</tr>
<tr>
<td>CX0025</td>
<td>Signature cannot be processed.</td>
</tr>
<tr>
<td>CX0026</td>
<td>Keystore cannot be processed.</td>
</tr>
<tr>
<td>CX0027</td>
<td>An I/O Exception occurred.</td>
</tr>
<tr>
<td>CX0028</td>
<td>The specified signature type is not supported. Added to the official specification.</td>
</tr>
</tbody>
</table>

---

**Changelog**

Version 9.3

Version 8.6

- Updated: `crypto:hmac`: The key can now be a string or a binary item.

The Module was introduced with Version 7.0.
Chapter 41. 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 and errors in this module are assigned to the http://basex.org/modules/csv namespace, which is statically bound to the csv prefix.

Conversion

XML: Direct, Attributes

If the direct or attributes format is chosen, a CSV string is converted to XML:

- 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:
  - If format is set to direct, the field names are encoded, as described in the Conversion Module, and used as element names.
  - Otherwise, if format is attributes, the field names will be stored in name attributes.

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.

XQuery

With the xquery format, CSV records are converted to a sequence of arrays:

- The resulting value will be a map with a records and an optional names key.
- Records are organized as a sequence of arrays. A single array contains the entries of a single record.
- The column names will be available if header option is set to true.

The CSV map can e.g. be accessed as follows:

- $csv?records[5] returns all entries of the 5th record (row)
- $csv?records?(2) returns all entries of the 2nd field (column)
- $csv?names?* returns the names of all fields (if available)
- Return enumerated strings for all records:

  for $record at $pos in $csv?records
  return $pos || "." || string-join($record?, ', ', '')

The resulting representation consumes less memory than XML-based formats, and values can be directly accessed without conversion. Thus, it is recommendable for very large inputs and for efficient ad-hoc processing.
## Options

In the following table, all available options are listed. The Excel column lists recommended options for data that is processed with Excel or Open/Libre Office.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
<th>Direction</th>
<th>Excel</th>
</tr>
</thead>
<tbody>
<tr>
<td>separator</td>
<td>Defines the character which separates the values of a single record.</td>
<td>comma, semicolon, colon, tab, space or a single character</td>
<td>comma</td>
<td>parse, serialize</td>
<td>semicolon or comma, depending on the region</td>
</tr>
<tr>
<td>header</td>
<td>Indicates if the first line of the parsed or serialized CSV data is a table header.</td>
<td>yes, no</td>
<td>no</td>
<td>parse, serialize</td>
<td></td>
</tr>
<tr>
<td>format</td>
<td>Specifies the format of the XML data:</td>
<td>direct, direct attributes, xquery</td>
<td>direct</td>
<td>parse, serialize</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• With direct conversion, field names are represented as element names</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• With attributes conversion, field names are stored in name attributes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• With xquery conversion, the input is converted to an XQuery map</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lax</td>
<td>Specifies if a lax approach is used to convert QNames to JSON names.</td>
<td>yes, no</td>
<td>yes</td>
<td>parse, serialize</td>
<td>no</td>
</tr>
<tr>
<td>quotes</td>
<td>Specifies how quotes are parsed:</td>
<td>yes, no</td>
<td>yes</td>
<td>parse, serialize</td>
<td>yes</td>
</tr>
<tr>
<td></td>
<td>• Parsing: If the option is enabled, quotes at the start and end of a value will be treated as control characters. Separators and</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
newlines within the quotes will be adopted without change.

- Serialization: If the option is enabled, the value will be wrapped with quotes if it contains characters that might be treated as control characters. A quote character in the value will be encoded according to the rules of the `backslashes` option.

```
backslashes
```

Specifies how quotes and other characters are escaped:

- Parsing: If the option is enabled, \r, \n and \t will be replaced with the corresponding control characters. All other escaped characters will be adopted as literals (e.g.: \" → "). If the option is disabled, two consecutive quotes will be replaced with a single quote (unless quotes is enabled and the quote is the first or last
character of a value).

- Serialization:
  If the option is enabled, \t, \r, \n, `, and the separator character will be encoded with a backslash. If the option is disabled, quotes will be duplicated.

---

allow
In Excel, a value will be evaluated if it starts with the character -, +, =, $, \t or \r. A regular expression can be specified to reject data that will be handled differently than expected by an application, or that may be malicious (see https://owasp.org/www-community/attacks/CSV_Injection for more details).

\string serialize\n[^-+=\t\r]*\[^-+]*|[^-+]?\d*\d+?

---

## Functions

### csv:doc

**Signatures**
csv:doc($uri as xs:string?) as item()?, csv:doc($uri as xs:string?, $options as map(*)?) as item()?

**Summary**
Fetches the CSV document referred to by the given $uri and converts it to an XQuery value. The $options argument can be used to control the way the input is converted.

**Errors**
parse: the specified input cannot be parsed as CSV document.options: the specified options are conflicting.

### csv:parse

**Signatures**
csv:parse($string as xs:string?) as item()?, csv:parse($string as xs:string?, $options as map(*)?) as item()?

**Summary**
Converts the CSV $string to an XQuery value. The $options argument can be used to control the way the input is converted.
Errors  parse: the specified input cannot be parsed as CSV document.

csv:serialize

Signatures  csv:serialize($input as item()? as xs:string, $options as map(*)?) as xs:string

Summary  Serializes the specified $input as CSV, using the specified $options, and returns the result as string. Values can also be serialized as CSV with the standard Serialization feature of XQuery:
  • The parameter method needs to be set to csv, and
  • the options presented in this article need to be assigned to the csv parameter.

Errors  serialize: 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 $options := map { 'lax': false() }
let $input := file:read-text('some-data.csv')
let $output := $input => csv:parse($options) => csv:serialize($options)
return $input eq $output
```

Example 3: Converts CSV data to XQuery and returns distinct column values:

Query:

```
let $text := `'[Name,City
  Jack,Chicago
  Jack,Washington
  John,New York
`'
let $options := map { 'format': 'xquery', 'header': true() }
let $csv := csv:parse($text, $options)
```
return (  
'Distinct values:',
  let $records := $csv('records')
  for $name at $pos in $csv('names')?*
  let $values := $records?($pos)
  return (  
    '* ' || $name || ': ' || string-join(distinct-values($values), ', ')
  )
)

Result:

Distinct values:
* Name: Jack, John
  * City: Chicago, Washington, New York

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse</td>
<td>The input cannot be parsed.</td>
</tr>
<tr>
<td>serialize</td>
<td>The node cannot be serialized.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.7
• Added: Options: allow option.

Version 9.4
• Added: csv:doc

Version 9.1
• Updated: csv:parse can be called with empty sequence.

Version 9.0
• Added: xquery option
• Removed: map option
• Updated: error codes updated; errors now use the module namespace

Version 8.6
• Updated: Options: improved Excel compatibility

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 42. Database Module

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 and errors in this module are assigned to the http://basex.org/modules/db namespace, which is statically bound to the db prefix.

Database Nodes

In BaseX, two internal representations exist for nodes.

- XML fragments are generated by XQuery node constructors.

- Database nodes are:
  - stored in a persistent database on disk;
  - nodes of a document that has been generated temporarily with fn:doc, fn:parse-xml and other functions; or
  - result of a main-memory update operation.

Some operations are restricted to database nodes, but you can convert XML fragments to database nodes by applying an empty update or transform operation to a node. Two examples:

- Retrieve the internal node id of an XML fragment:
  let $xml := <xml>hello world</xml> update {}
  return db:node-id($xml/text())

- Puts a marker element around the result of a full-text request (see ft:mark for more details):
  copy $p := <xml>hello world</xml>
  modify ()
  return ft:mark($p[=text() contains text 'word'], 'b')

Updating Functions

Various functions in this module are updating. Updating functions will not be immediately executed, but queued on the Pending Update List, and processed after the remaining query has been evaluated. This means that the order in which the functions are specified in the query often does not reflect the order in which they will eventually be executed.

General Functions

db:system

<table>
<thead>
<tr>
<th>Signatures</th>
<th>db:system() as element(system)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns general information on the database system the current values of all global and local Options. The INFO command returns similar output.</td>
</tr>
</tbody>
</table>

db:option

<table>
<thead>
<tr>
<th>Signatures</th>
<th>db:option($name as xs:string) as xs:string</th>
</tr>
</thead>
</table>
Summary
Returns the current value (string, integer, boolean, map) of a global or local Option with the specified $name. The SHOW OPTIONS command returns similar output.

Errors
option: the specified option is unknown.

Examples
- `db:option('dbpath')` returns the database path string.
- `db:option('serializer')` returns a map with the current serialization parameters.
- `declare option db:stripws 'true'; db:option('stripws')` returns the locally assigned value.

---

Summary
Returns the value (string, boolean, integer) of a property with the specified $name in the database $db. The available properties are the ones returned by `db:info`.

Errors
property: the specified property is unknown.

Examples
- `db:property('db', 'size')` returns the number of bytes occupied by the database db.
- `db:property('xmark', 'textindex')` indicates if the xmark database has a text index.
- `db:property('discogs', 'uptodate')` indicates if the database statistics and index structures of the discogs database are up-to-date.

---

Summary
The result of this function is dependent on the number of arguments:
- Without arguments, the names of all databases are returned that are accessible to the current user.
- If a database $db is specified, all documents and raw files of the specified database are returned.
- The list of returned resources can be restricted by the $path argument.

Errors
open: the addressed database does not exist or could not be opened.

Examples
- `db:list("docs")` returns the names of all documents of a database named docs.

---

Summary
Without arguments, an element is returned for each database that is accessible to the current user:
- An element has a value, which is the name of the database, and several attributes, which contain the number of stored resources, the modification date, the database size on disk (measured in bytes), and a path to the original database input.
If a database $\text{db}$ is specified, an element for each document and raw file of the specified database is returned:

- An element has a value, which is the name of the resource, and several attributes, which contain the content type, the modification date, the raw flag (which indicates if the resource is binary or XML), and the size of a resource.

- The value of the size attribute depends on the resource type: for documents, it represents the number of nodes; for binary data, it represents the file size (measured in bytes).

- Returned databases resources can be further restricted by the $\text{path}$ argument.

**Errors**

open: the addressed database does not exist or could not be opened.

**Examples**

- $\text{db:list-details("shop")}$ returns the names plus additional info on all resources of a database named shop.

### db:dir

**Signatures**

$\text{db:dir($db as xs:string, $path as xs:string) as element()}$

**Summary**

Returns metadata on all directories and resources of the database $\text{db}$ in the specified directory $\text{path}$. Two types of elements are returned:

- resource represents a resource. The element value is the directory path; content type, modification date, raw flag (which indicates if the resource is binary or XML), and size of the resource are returned as attributes.

- dir represents a directory. The element value is the directory path; the modification date is returned as attribute.

Please note that directories are not stored in BaseX. Instead, they result implicitly from the paths of stored resources.

**Errors**

open: the addressed database does not exist or could not be opened.

**Examples**

- $\text{db:dir('shop', 'books')}$ returns all entries of the books directory of a shop database.

### Read Operations

### db:get

*Updated with Version 10: Renamed (before: db:open). Due to its widespread use, the old function name will be supported for some more time.*

**Signatures**

$\text{db:get($db as xs:string) as document-node()}$,

$\text{db:get($db as xs:string, $path as xs:string) as document-node()}$

**Summary**

Returns all documents from the database $\text{db}$, or only documents matching the specified $\text{path}$.

**Errors**

open: the addressed database does not exist or could not be opened.

**Examples**

- $\text{db:get('docs')}$ returns all documents from the database named docs.

- $\text{db:get('db', 'one')}$ returns all documents from the database named db located in the path one.

- for $i$ in 1 to 3 return $\text{db:get('db' | | $i)}$/item returns all item elements from the databases db1, db2 and db3.

### db:get-pre

*Updated with Version 10: Renamed (before: db:open-pre).*
### Database Module

#### Signatures

**db:get-pre**($db as xs:string, $pres as xs:integer*) as node()*

#### Summary

Returns all nodes from the database $db with the pre values $pres in distinct document order. 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

- **open**: the addressed database does not exist or could not be opened.
- **range**: the specified pre value does not exist in the database.

#### Examples

- `db:get-pre("docs", 0)` returns the first database node from the database named docs.

### db:get-id

*Updated with Version 10:* Renamed (before: open-id).

#### Signatures

**db:get-id**($db as xs:string, $ids as xs:integer*) as node()*

#### Summary

Returns all nodes from the database $db with the pre values $ids in distinct document order. Each database node has a persistent ID value. Access to the node ID can be sped up by turning on the UPDINDEX option.

#### Errors

- **open**: the addressed database does not exist or could not be opened.
- **range**: the specified ID value does not exist in the database.

### db:get-binary

*Updated with Version 10:* renamed (before: db:retrieve).

#### Signatures

**db:get-binary**($db as xs:string, $path as xs:string) as item()

#### Summary

Returns a map with all paths and binary resources of the database $db. A single xs:base64Binary item is returned if a $path is specified. All items are lazy, i.e., the actual data will only be retrieved if it is processed.

#### Errors

- **open**: the addressed database does not exist or could not be opened.
- **mainmem**: the database is not persistent (stored on disk).

#### Examples

- `db:get-binary('DB', 'music/01.mp3')` returns the specified audio file as raw data.
- `stream:materialize(db:get-binary('DB', 'music/01.mp3'))` materializes the streamable result in main-memory before returning it.
- `convert:binary-to-string(db:get-binary('DB', 'info.txt'), 'UTF-8')` converts a binary database resource as UTF-8 text and returns a string.

### db:get-value

*Introduced with Version 10.*

#### Signatures

**db:get-value**($db as xs:string, $path as xs:string) as item()*

#### Summary

Returns a map with all paths and values of the database $db. A single value is returned if a $path is specified.

#### Errors

- **open**: the addressed database does not exist or could not be opened.
- **mainmem**: the database is not persistent (stored on disk).

#### Examples

- `db:get-value('DB', 'sequence')` returns the specified sequence.

### db:node-pre

#### Signatures

**db:node-pre**($nodes as node()*) as xs:integer*

#### Summary

Returns the pre values of the specified $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  node: $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 specified $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  node: $nodes contains a node which is not stored in a database.

**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. As with fn:serialize(), the parameters can be specified

• either as children of an <output:serialization-parameters/> element:

```
<output:serialization-parameters>
  <output:method value='xml'/>
  <output:cdata-section-elements value="div"/>...
</output:serialization-parameters>
```

• or as map, which contains all key/value pairs:

```
map { "method": "xml", "cdata-section-elements": "div", ... }
```

Errors  open: 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 code can be used to export parts of the database:

```
let $target := '/home/john/xml/target'
for $doc in db:get('DB', 'collection')
let $path := $target || db:path($doc)
return (file:create-dir(file:parent($path)),
        file:write($path, $doc))
```

**Value Indexes**

**db:text**

**Signatures**  db:text($db as xs:string, $strings as xs:string*) as text()*

**Summary**  Returns all text nodes of the database $db that have one of the specified $strings as values and that are stored in the text index.

Errors  open: the addressed database does not exist or could not be opened. no-index: the index is not available.
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 whose values are between $min and $max and that are stored in the text index.

Errors:
- open: the addressed database does not exist or could not be opened.
- no-index: the index is not available.

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, $strings as xs:string*) as attribute()*, db:attribute($db as xs:string, $strings as xs:string*, $name as xs:string) as attribute()*
```

Summary: Returns all attribute nodes of the database $db that have one of the specified $strings as values and that are stored in the attribute index. If $name is specified, the resulting attribute nodes are filtered by their attribute name.

Errors:
- open: the addressed database does not exist or could not be opened.
- no-index: the index is not available.

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, $name 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 and that are stored in the attribute index.

Errors:
- open: the addressed database does not exist or could not be opened.
- no-index: the index is not available.

Examples:
* db:attribute-range("DB", "id456", "id473", 'id') returns all @id attributes of the database DB that have a string value in between id456 and id473.

**db:token**

Signatures: 
```
db:token($db as xs:string, $tokens as xs:string*) as attribute()*, db:token($db as xs:string, $tokens as xs:string*, $name as xs:string) as attribute()*
```

Summary: Returns all attribute nodes of the database $db the values of which contain one of the specified $tokens. If $name is specified, the resulting attribute nodes are filtered by their attribute name.

Errors:
- open: the addressed database does not exist or could not be opened.
- no-index: the index is not available.

Examples:
* db:token("DB", "row", "class").parent::div returns all div nodes of database DB with a class attribute that contains the token row.
Updates

All functions in this section are Updating Functions.

**db:create**

**Signatures**

\[
db:create($db \text{ as } xs:string) \text{ as } empty-sequence(),
\]

\[
db:create($db \text{ as } xs:string, \ $inputs \text{ as item()}*) \text{ as } empty-sequence(),
\]

\[
db:create($db \text{ as } xs:string, \ $inputs \text{ as item()}*, \ $paths \text{ as } xs:string*) \text{ as } empty-sequence(),
\]

\[
db:create($db \text{ as } xs:string, \ $inputs \text{ as item()}*, \ $paths \text{ as } xs:string*, \ $options \text{ as } map(*)?) \text{ as } empty-sequence()
\]

**Summary**

Creates a new database with name $db and adds initial documents specified via $inputs to the specified $paths:

- $inputs may be strings or nodes:
  - nodes may be of any type except for attributes
  - strings can be a URI pointing to a file/directory or an XML string (which is detected by the leading < character)
  - a path must be specified if the input is not a file or directory reference
- The parsing and indexing behavior can be controlled via $options:
  - allowed options are ADDCACHE and the indexing, full-text indexing, parsing and XML parsing options, all in lower case
  - parsing options will only impact string input (URIs, XML strings), because nodes have already been parsed.
- An existing database will be overwritten.
- Database creation takes place after most other update operations (see Pending Update List). As a consequence, a newly created database cannot be addressed in the same query.

**Errors**

- lock: a database is opened by another process
- name: the specified name is not a valid database name
- conflict: the same database was addressed more than once
- args: 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, preserving relative filesystem paths and creating a full-text index.

**db:add**

**Signatures**

\[
db:add($db \text{ as } xs:string, \ $input \text{ as item()}) \text{ as } empty-sequence(),
\]

\[
db:add($db \text{ as } xs:string, \ $input \text{ as item()}, \ $path \text{ as } xs:string?) \text{ as } empty-sequence(),
\]

\[
db:add($db \text{ as } xs:string, \ $input \text{ as item()}, \ $path \text{ as } xs:string?, \ $options \text{ as } map(*)?) \text{ as } empty-sequence()
\]
Summary

Adds documents specified by $input to the database $db with the specified $path:

- A document with the same path may occur more than once in a database. If you want to enforce single instances, use db:put instead.
- See db:create for more details on the input and path arguments.
- The parsing behavior can be controlled via $options:
  - allowed options are ADDCACHE and the parsing and XML parsing options, all in lower case
  - parsing options will only impact string input (URIs, XML strings), because nodes have already been parsed

Errors

open: the addressed database does not exist or could not be opened.

db:put

Updated with Version 10: renamed (before: db:replace); function signature aligned with db:add (second and third argument swapped).

Signatures
db:put($db as xs:string, $input as item(), $path as xs:string) as empty-sequence(), db:put($db as xs:string, $input as item(), $path as xs:string, $options as map(*)?) as empty-sequence()

Summary

Replaces a resource, specified by $path, in the database $db with the contents of $input, or adds it as a new resource:

- The parsing behavior can be controlled via $options:
  - Allowed options are ADDCACHE and the parsing and XML parsing options, all in lower case.
  - Parsing options will only impact string input (URIs, XML strings), because nodes have already been parsed.
  - See db:create for more details on the input argument.

Errors

open: the addressed database does not exist or could not be opened.
target: the path points to a directory.

Examples

- db:put("DB", "/home/dir/doc.xml", "docs/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:put("DB", "<a/>", "docs/dir/doc.xml") replaces the content of the document docs/dir/doc.xml in the database DB with <a/>.
- db:put("DB", document { <a/> }, "docs/dir/doc.xml") 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:put('db', doc($path), $file)

**db:put-binary**

*Updated with Version 10*: renamed (before: db:put); function signature aligned with db:add (second and third argument swapped).

**Signatures**

\[
db:put-binary($db as xs:string, $input as item(), $path as xs:string) as empty-sequence()
\]

**Summary**

Stores a binary resource specified by $input in the database $db at the specified $path. Existing resources are overwritten.

**Errors**

- open: the addressed database does not exist or could not be opened.
- mainmem: the database is not persistent (stored on disk).

**Examples**

- `db:put-binary('DB', file:read-binary('video.mov'), 'video/sample.mov')` stores the addressed video file at the specified location.

- With the following query, you can copy the binary resources of one database into another:

```
let $db := 'db'
let $src-path := 'src/
let $trg-path := 'trg/
for $src in db:list($db, $src-path)
where db:type($db, $src) = 'binary'
let $trg := $trg-path || substring-after($src, $src-path)
return db:put-binary($db, db:get-binary($db, $src), $trg)
```

**db:put-value**

*Introduced with Version 10.*

**Signatures**

\[
\]

**Summary**

Stores a value specified by $input in the database $db at the specified $path. Existing resources are overwritten. The value can be an arbitrary sequence of atomic items, nodes, maps, and arrays.

**Errors**

- open: the addressed database does not exist or could not be opened.
- mainmem: the database is not persistent (stored on disk).

**Examples**

- `db:put-value('DB', 1 to 10000, 'sequence')` stores a numeric range in the database.

- With the following query, a map with countries and associated cities is stored in a database. The value resource can e.g. be used as index in future queries:

```
db:put-value(
    'factbook',
    map:merge(
        for $country in db:get('factbook')//country
        return map:entry($country/@name, $country//city/name ! string())
    ),
    'cities'
)
```

**db:delete**

**Signatures**

\[
db:delete($db as xs:string, $path as xs:string) as empty-sequence()
\]
### Database Module

<table>
<thead>
<tr>
<th><strong>Summary</strong></th>
<th>Deletes resource(s), specified by $path, from the database $db.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Errors</strong></td>
<td>open: the addressed database does not exist or could not be opened. path: the specified path is invalid.</td>
</tr>
</tbody>
</table>
| **Examples**| • db:delete("DB", "docs/dir/doc.xml") deletes the resource docs/dir/doc.xml from DB.  
• db:delete("DB", "docs/dir") deletes all resources from DB in the specified path docs/dir. |

#### db:copy

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>db:copy($db as xs:string, $name as xs:string) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Creates a copy of the database $db, which will be called $name.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>open: the addressed database does not exist or could not be opened. lock: a database is opened by another process. name: invalid database name. conflict: the same database was addressed more than once.</td>
</tr>
</tbody>
</table>

#### db:alter

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>db:alter($db as xs:string, $name as xs:string) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Renames the database $db to $name.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>open: the addressed database does not exist or could not be opened. lock: a database is opened by another process. name: invalid database name. conflict: the same database was addressed more than once.</td>
</tr>
</tbody>
</table>

#### db:optimize

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>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 map(*)?) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Optimizes the metadata 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 supported if $all is true.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>open: the addressed database does not exist or could not be opened.</td>
</tr>
</tbody>
</table>
| **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

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>db:rename($db as xs:string, $source as xs:string, $target as xs:string) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Moves all resource(s) of database $db, which are found in the supplied $source path, to the supplied $target path. The paths may point to single resources or directories. No updates will take place if a non-existing source path is supplied.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>open: the addressed database does not exist or could not be opened. path: the specified source or target path, or one of its descendants, is invalid.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>• db:rename(&quot;DB&quot;, &quot;docs/dir/doc.xml&quot;, &quot;docs/dir/newdoc.xml&quot;) renames the resource docs/dir/doc.xml to docs/dir/newdoc.xml in the database DB.</td>
</tr>
</tbody>
</table>
Database Module

- \texttt{db:rename}("DB", "docs/dir", "docs/newdir") moves all resources in the database DB from docs/dir to \{Code|docs/newdir\}.

\textbf{db:flush}

\begin{tabular}{|l|}
\hline
\end{tabular}

Signatures \texttt{db:flush($\text{db}$ as xs:string) as empty-sequence()}

Summary Explicitly flushes the buffers of the database $\text{db}$. This command is only useful if \texttt{AUTOFLUSH} has been set to false.

Errors open: the addressed database does not exist or could not be opened.

\textbf{db:drop}

\begin{tabular}{|l|}
\hline
\end{tabular}

Signatures \texttt{db:drop($\text{db}$ as xs:string) as empty-sequence()}

Summary Drops the database $\text{db}$ and all connected resources.

Errors open: the addressed database does not exist or could not be opened. lock: a database is opened by another process. conflict: the same database was addressed more than once.

Examples • \texttt{db:drop("DB")} drops the database DB.

\section*{Backups}

\textit{Introduced with Version 10}: Support for general data (registered users, scheduled services and key-value stores).

All functions in this section except for \texttt{db:backups} are Updating Functions.

\textbf{db:create-backup}

\textit{Updated with Version 10}: Options argument added.

\begin{tabular}{|l|}
\hline
\end{tabular}

Signatures \texttt{db:create-backup($\text{db}$ as xs:string) as empty-sequence()}, \texttt{db:create-backup($\text{db}$ as xs:string, $\text{options}$ as map(*)) as empty-sequence()}

Summary Creates a backup of the database $\text{db}$. If no name is supplied, general data will be backed up. The following $\text{options}$ are available:

- With \texttt{comment}, a comment string can be attached to the backup.
- By setting \texttt{compress} to false, the backup will be created faster, but it will take more space on disk.

Errors open: the addressed database does not exist or could not be opened. name: invalid database name. conflict: the same database was addressed more than once.

Examples • \texttt{db:create-backup('DB', map { 'compress': false() })} creates a backup of the database DB without compressing its entries.

\textbf{db:drop-backup}

\begin{tabular}{|l|}
\hline
\end{tabular}

Signatures \texttt{db:drop-backup($\text{name}$ as xs:string) as empty-sequence()}

Summary Drops all backups of the database with the specified $\text{name}$. If the name ends with a timestamp, only the specified backup file will be deleted. If no name is supplied, backups with general data are addressed.

Errors backup: No backup file found. name: invalid database name. conflict: the same database was addressed more than once.

Examples • \texttt{db:drop-backup("DB")} drops all backups of the database DB.

- \texttt{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:alter-backup

Signatures | db:alter-backup($name as xs:string, $new-name as xs:string) as empty-sequence()
---|---
Summary | Renames all backups of the database with the specified $name to $new-name. If the name ends with a date, only the specified backup file will be renamed.
Errors | backup: No backup file found.name: invalid database name.conflict: the same database was addressed more than once.
Examples | • db:alter-backup("DB", "DB2") renames all backups of the database DB to DB2.

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. If no name is supplied, general data will be restored. If general data is restored, it will only be available after BaseX has been restarted.
Errors | lock: a database is opened by another process.name: invalid database name.no-backup: No backup found.conflict: the same database was addressed more than once.
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: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 with timestamp, file size and comment. 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.

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 | node: $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 | node: $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 $\text{db}$ or the resource specified by $\text{path}$ 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:type**

*Introduced with BaseX 10: Replaces `db:is-raw` and `db:is-xml`.*

**Signatures**

db:type($\text{db}$ as xs:string, $\text{path}$ as xs:string) as xs:boolean

**Summary**
Returns the type of a resource – xml, binary, or value – in the database $\text{db}$ at the specified $\text{path}$.

**Errors**
open: the addressed database does not exist or could not be opened.

**Examples**
- `db:type("DB", "factbook.xml")` returns true if the specified resource is an XML document.

**db:content-type**

**Signatures**

db:content-type($\text{db}$ as xs:string, $\text{path}$ as xs:string) as xs:string

**Summary**
Retrieves the content type of a resource in the database $\text{db}$ and the path $\text{path}$. 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**
open: the addressed database does not exist or could not be opened.

**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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>args</td>
<td>The number of specified inputs and paths differs.</td>
</tr>
<tr>
<td>conflict</td>
<td>Multiple update operations point to the same target.</td>
</tr>
<tr>
<td>lock</td>
<td>A database cannot be updated because it is opened by another process.</td>
</tr>
<tr>
<td>mainmem</td>
<td>The addressed database is not persistent (stored on disk).</td>
</tr>
<tr>
<td>name</td>
<td>The name of the specified database is invalid.</td>
</tr>
<tr>
<td>no-backup</td>
<td>No backup exists for a database.</td>
</tr>
<tr>
<td>node</td>
<td>The referenced XML node is no database node, i.e. it is neither stored in a database nor represented as database fragment.</td>
</tr>
<tr>
<td>no-index</td>
<td>The database lacks an index structure required by the called function.</td>
</tr>
<tr>
<td>open</td>
<td>The addressed database does not exist or could not be opened.</td>
</tr>
<tr>
<td>option</td>
<td>The specified option is unknown.</td>
</tr>
<tr>
<td>path</td>
<td>The specified database path is invalid.</td>
</tr>
</tbody>
</table>

247
Database Module

property The specified database property is unknown.
range The addressed database id or pre value is out of range.
target Path points to an invalid target.

Changelog

Version 10
• Added: db:get, db:put, db:type.
• Added: Backups: Support for general data (registered users, scheduled services and key-value stores).
• Updated: db:get, db:get-id, db:get-pre renamed (before: db:open, db:open-id, db:open-pre)
• Updated: db:put renamed (before: db:replace); function signature aligned with db:add (second and third argument swapped).
• Updated: db:put-binary renamed (before: db:store); function signature aligned with db:add (second and third argument swapped).
• Updated: db:get-binary renamed (before: db:retrieve).
• Updated: db:backups, db:create-backup: Options added.
• Removed: db:is-raw, db:is-raw (new: db:type).

Version 9.3
• Added: db:alter-backup
• Updated: db:open-id, db:open-pre: support for multiple integers

Version 9.2
• Added: db:dir
• Updated: db:add: $path allow empty path argument

Version 9.0
• Added: db:option
• Updated: db:output renamed to update:output, db:output-cache renamed to update:cache
• Updated: error codes updated; errors now use the module namespace

Version 8.6
• Added: db:property

Version 8.4
• Updated: db:create, db:add, db:replace: support for ADDCACHE option.
• Added: db:token

Version 8.3
• Updated: db:list-details: attributes with name of database and date of backup added to results.
• Updated: db:backups now include attributes with name of database and date of backup.
• Updated: Value Indexes: raise error if no index exists.

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:exists, db:retrieve, db:store, db:is-raw, db:is-xml
• Updated: db:list, db:open, db:add
Chapter 43. 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 Client Module can be used. With the HTML Module, retrieved HTML documents can be converted to XML.

Conventions

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

URI arguments can point be URLs or point to local files. Relative file paths will be resolved against the current working directory (for more details, have a look at the File Module).

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 lazy xs:base64Binary item.

Errors

open: the URI could not be resolved, or the resource could not be retrieved.

Examples


• lazy:cache(fetch:binary("http://en.wikipedia.org")) enforces the fetch operation (otherwise, it will be delayed until requested first).

fetch:text

Signatures

fetch:text($uri as xs:string) as xs:string,
fetch:text($uri as xs:string, $encoding as xs:string) as xs:string,
fetch:text($uri as xs:string, $encoding as xs:string, $fallback as xs:boolean) as xs:string.

Summary

Fetches the resource referred to by the given $uri and returns it as lazy xs:string item:

• The UTF-8 default encoding can be overwritten with the optional $encoding argument.

• By default, invalid characters will be rejected. If $fallback is set to true, these characters will be replaced with the Unicode replacement character FFFD (#).

Errors

open: the URI could not be resolved, or the resource could not be retrieved.
encoding: the specified encoding is not supported, or unknown.

Examples


• lazy:cache(fetch:text("http://en.wikipedia.org")) enforces the fetch operation (otherwise, it will be delayed until requested first).
**fetch:doc**

**Signatures**
fetch:doc($uri as xs:string) as document-node(),
fetch:doc($uri as xs:string, $options as map(*)?) as document-node()

**Summary**
Fetches the resource referred to by the given $uri and returns it as a document node. The $options argument can be used to change the parsing behavior. Allowed options are all parsing and XML parsing options in lower case. The function differs from fn:doc in various aspects:

- It is **non-deterministic**, i.e., a new document node will be created by each call of this function.
- A document created by this function will be garbage-collected as soon as it is not referenced anymore.
- URIs will not be resolved against existing databases. As a result, it will not trigger any locks (see limitations of database locking for more details).

**Errors**
open: the URI could not be resolved, or the resource could not be retrieved.

**Examples**
- Retrieve an XML representation of the English Wikipedia main HTML page with whitespace stripped:
  ```xml
  fetch:doc("http://en.wikipedia.org", map { 'stripws': true() })
  ```
- Return a web page as XML, preserve namespaces:
  ```xml
  fetch:doc(
    'http://basex.org/',
    map {
      'parser': 'html',
      'htmlparser': map { 'nons': false() }
    }
  )
  ```

**fetch:binary-doc**

**Signatures**
fetch:binary-doc($input as xs:anyAtomicType) as document-node(),
fetch:binary-doc($data as xs:anyAtomicType, $options as map(*)?) as document-node()

**Summary**
Converts the specified $input (xs:base64Binary, xs:hexBinary) to XML and returns it as a document node. In contrast to fn:parse-xml, which expects a string, the input can be arbitrarily encoded. The encoding will be derived from the XML declaration or (in case of UTF-16 or UTF-32) from the first bytes of the input. The $options argument can be used to change the parsing behavior. Allowed options are all parsing and XML parsing options in lower case.

**Examples**
- Retrieves file input as binary data and parses it as XML:
  ```xml
  fetch:binary-doc(file:read-binary('doc.xml'))
  ```
- Encodes a string as CP1252 and parses it as XML. The input and the string `touché` will be correctly decoded because of the XML declaration:
  ```xml
  fetch:binary-doc(convert:string-to-base64("<?xml version='1.0' encoding='CP1252'?><xml>touché</xml>", "CP1252")
  ```
- Encodes a string as UTF-16 and parses it as XML. The document will be correctly decoded, as the first bytes of the data indicate that the input must be UTF-16:
  ```xml
  fetch:binary-doc(convert:string-to-base64("<xml/>", "UTF16")
  ```

**Errors**
open: the input could not be parsed.
**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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>The specified encoding is not supported, or unknown.</td>
</tr>
<tr>
<td>open</td>
<td>The URI could not be resolved, or the resource could not be retrieved.</td>
</tr>
</tbody>
</table>

**Examples**


## Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoding</td>
<td>The specified encoding is not supported, or unknown.</td>
</tr>
<tr>
<td>open</td>
<td>The URI could not be resolved, or the resource could not be retrieved.</td>
</tr>
</tbody>
</table>

## Changelog

**Version 10.0**


**Version 9.0**

- Added: `fetch:xml-binary`
- Updated: error codes updated; errors now use the module namespace

**Version 8.5**

- Updated: `fetch:text:$fallback` argument added.

**Version 8.0**

- Added: `fetch:xml`

The module was introduced with Version 7.6.
Chapter 44. 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. The following enhancements have not been added to the specification yet:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file:descendants</td>
<td>new function</td>
</tr>
<tr>
<td>file:is-absolute</td>
<td>new function</td>
</tr>
<tr>
<td>file:read-text, file:read-text-lines</td>
<td>$fallback argument added</td>
</tr>
<tr>
<td>file:read-text-lines</td>
<td>$offset and $length arguments added</td>
</tr>
<tr>
<td>file:resolve-path</td>
<td>$base argument added</td>
</tr>
</tbody>
</table>

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.

The error invalid-path is raised if a path is invalid.

File Paths

• All file paths are resolved against the current working directory (the directory from which BaseX or, more generally, the Java Virtual Machine, was started). This directory can be retrieved via file:base-dir.

• A path can be specified as local filesystem path or as file URI.

• Returned strings that refer to existing directories are suffixed with a directory separator.

Read Operations

file:list

Signatures

<table>
<thead>
<tr>
<th>file:list($dir as xs:string) as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>file:list($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*</td>
</tr>
</tbody>
</table>

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 subdirectories 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* |
### File Module

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the full paths to all files and directories found in the specified <code>$dir</code>. The inverse function <code>file:parent</code> returns the full paths to directories. The related function <code>file:list</code> returns relative file paths.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td><code>not-found</code>: the specified file does not exist. <code>no-dir</code>: the specified path does not point to a directory. <code>io-error</code>: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

#### file:descendants

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>file:descendants($dir as xs:string) as xs:string*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the full paths to all files and directories found in the specified <code>$dir</code> and its subdirectories. The related function <code>file:list</code> returns relative file paths.</td>
</tr>
<tr>
<td>Errors</td>
<td><code>not-found</code>: the specified file does not exist. <code>no-dir</code>: the specified path does not point to a directory. <code>io-error</code>: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

#### file:read-binary

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>file:read-binary($path as xs:string) as xs:base64Binary</code>, <code>file:read-binary($path as xs:string, $offset as xs:integer) as xs:base64Binary</code>, <code>file:read-binary($path as xs:string, $offset as xs:integer, $length as xs:integer) as xs:base64Binary</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Reads the binary content of the file specified by <code>$path</code> and returns it as a lazy <code>xs:base64Binary</code> item. The optional parameters <code>$offset</code> and <code>$length</code> can be used to read chunks of a file.</td>
</tr>
<tr>
<td>Errors</td>
<td><code>not-found</code>: the specified file does not exist. <code>is-dir</code>: the specified path is a directory. <code>out-of-range</code>: the offset or length is negative, or the chosen values would exceed the file bounds. <code>io-error</code>: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**Examples**

- `lazy:cache(file:read-binary("config.data"))` enforces the file access (otherwise, it will be delayed until requested first).

#### file:read-text

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>file:read-text($path as xs:string) as xs:string</code>, <code>file:read-text($path as xs:string, $encoding as xs:string) as xs:string</code>, <code>file:read-text($path as xs:string, $encoding as xs:string, $fallback as xs:boolean) as xs:string</code></th>
</tr>
</thead>
</table>
| Summary | Reads the textual contents of the file specified by `$path` and returns it as a lazy `xs:string` item:
- The UTF-8 default encoding can be overwritten with the optional `$encoding` argument.
- By default, invalid characters will be rejected. If `$fallback` is set to true, these characters will be replaced with the Unicode replacement character FFFD (#). |
| 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. |

**Examples**

- `lazy:cache(file:read-text("ids.txt"))` enforces the file access (otherwise, it will be delayed until requested first).

#### file:read-text-lines

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>file:read-text-lines($path as xs:string) as xs:string*</code>, <code>file:read-text-lines($path as xs:string, $encoding as xs:string) as xs:string*</code>, <code>file:read-text-lines($path as xs:string, $encoding as xs:string, $fallback as xs:boolean) as xs:string*</code>, <code>file:read-text-lines($path as xs:string, $encoding as xs:string, $fallback as xs:boolean, $offset as xs:integer) as xs:string*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>The UTF-8 default encoding can be overwritten with the optional <code>$encoding</code> argument. By default, invalid characters will be rejected. If <code>$fallback</code> is set to true, these characters will be replaced with the Unicode replacement character FFFD (#).</td>
</tr>
<tr>
<td>Errors</td>
<td><code>not-found</code>: the specified file does not exist. <code>is-dir</code>: the specified path is a directory. <code>unknown-encoding</code>: the specified encoding is not supported, or unknown. <code>io-error</code>: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**Examples**

- `lazy:cache(file:read-text-lines("ids.txt"))` enforces the file access (otherwise, it will be delayed until requested first).
lines($path as xs:string, $encoding as xs:string, $fallback as xs:boolean, $offset as xs:integer, $length as xs:integer) 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 UTF-8 default encoding can be overwritten with the optional $encoding argument.
- By default, invalid characters will be rejected. If $fallback is set to true, these characters will be replaced with the Unicode replacement character FFFD (#).

The lines to be read can be restricted with the optional parameters $offset and $length.

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 if it does not already exist. Non-existing parent directories will be created as well.

Errors
exists: the specified target exists, but is no directory.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 subdirectories 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. As with fn:serialize(), the parameters can be specified

- either as children of an `<output:serialization-parameters/>` element:

  ```
  <output:serialization-parameters>
  <output:method value='xml'/>
  <output:cdata-section-elements value='div'/>
  ...
  </output:serialization-parameters>
  ```

- or 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.
- out-of-range: the offset is negative, or it exceeds the current file size.
- io-error: the operation fails for some other reason.

**Examples**

- `file:write('data.bin', xs:hexBinary('414243'))` writes a hex representation to the specified file.
- `file:write('data.bin', xs:hexBinary('414243'), map { 'method': 'base64' })` writes binary data to the specified file (see Serialization for more details).

**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 exist, 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 exist, 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 `$path`. If the specified file does not exist, a new file is created. The optional parameter `$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 exist, 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 Module

file:copy

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:copy($source as xs:string, $target as xs:string) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

file:move

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:move($source as xs:string, $target as xs:string) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

File Properties

file:exists

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:exists($path as xs:string) as xs:boolean,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns an xs:boolean indicating whether a file or directory specified by $path exists in the file system.</td>
</tr>
</tbody>
</table>

file:is-dir

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:is-dir($path as xs:string) as xs:boolean,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns an xs:boolean indicating whether the argument $path points to an existing directory.</td>
</tr>
</tbody>
</table>

file:is-absolute

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:is-absolute($path as xs:string) as xs:boolean,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

file:is-file

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:is-file($path as xs:string) as xs:boolean,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns an xs:boolean indicating whether the argument $path points to an existing file.</td>
</tr>
</tbody>
</table>

file:last-modified

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:last-modified($path as xs:string) as xs:dateTime,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Retrieves the timestamp of the last modification of the file or directory specified by $path.</td>
</tr>
</tbody>
</table>
Errors | not-found: the specified path does not exist.

**file:size**

**Signatures** | file:size($path 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**

**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.

**Examples** | The following examples apply to Windows:

• file:resolve-path('file.txt', 'C:/Temp/') returns C:/Temp/file.txt.
• file:resolve-path('file.txt', 'C:/Temp') returns C:/file.txt.
• file:resolve-path('file.txt', 'Temp') raises an error.

**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 \r, \n, or \t.

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("").

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exists</td>
<td>A file with the same path already exists.</td>
</tr>
<tr>
<td>invalid-path</td>
<td>A specified path is invalid.</td>
</tr>
<tr>
<td>io-error</td>
<td>The operation fails for some other reason specific to the operating system.</td>
</tr>
<tr>
<td>is-dir</td>
<td>The specified path is a directory.</td>
</tr>
<tr>
<td>is-relative</td>
<td>The specified path is relative (and must be absolute).</td>
</tr>
<tr>
<td>no-dir</td>
<td>The specified path does not point to a directory.</td>
</tr>
<tr>
<td>not-found</td>
<td>A specified path does not exist.</td>
</tr>
<tr>
<td>out-of-range</td>
<td>The specified offset or length is negative, or the chosen values would exceed the file bounds.</td>
</tr>
<tr>
<td>unknown-encoding</td>
<td>The specified encoding is not supported, or unknown.</td>
</tr>
</tbody>
</table>
Changelog

Version 9.3
• Added: file:descendants

Version 9.0
• Updated: file:read-text-lines: $offset and $length arguments added.

Version 8.5
• Updated: file:read-text, file:read-text-lines: $fallback argument added.

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
• 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 subdirectories from being accidentally deleted.
• Fixed: file:list now returns relative instead of absolute paths.
Chapter 45. Full-Text Module

Read this entry online in the BaseX Wiki.

This XQuery Module extends the Full-Text features of BaseX: 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 and errors in this module are assigned to the http://basex.org/modules/ft namespace, which is statically bound to the ft prefix.

Database 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 map(*)?) 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. The following options are supported (the introduction on Full-Text processing gives you equivalent expressions in the XQuery Full-Text notation):

- **mode**: determine the mode how tokens are searched. Allowed values are any, any word, all, all words, and phrase. any is the default search mode.

- **wildcards**: turn wildcard querying on or off. Allowed values are true and false. By default, wildcard querying is turned off.

- **fuzzy**: turn fuzzy querying on or off. Allowed values are true and false. By default, fuzzy querying is turned off.

- **errors**: control the maximum number of tolerated errors for fuzzy querying. By default, 0 is assigned (see Fuzzy Querying for more details).

- **ordered**: indicate if all tokens must occur in the order in which they are specified. Allowed values are true and false. The default is false.

- **content**: specify 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**: define 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**: set up a window in which all tokens must be located. By default, the option is turned off. It has following sub options:

    - **size**: specify the size of the window in terms of units.
• unit: can be sentences, sentences or paragraphs. The default is words.

• distance: specify the distance in which tokens must occur. By default, the option is turned off. It has following sub options:
  • min: specify the minimum distance in terms of units. The default is 0.
  • max: specify the maximum distance in terms of units. The default is #.
  • unit: can be words, sentences or paragraphs. The default is words.

**Errors**
- db:get: The addressed database does not exist or could not be opened.
- db:no-index: the index is not available.
- options: 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": map {
      "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()
  for $db in 1 to 3
    let $dbname := 'DB' || $db
    return ft:search($dbname, $terms, map { 'fuzzy': $fuzzy })/..
  ```

### 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**
- db:get: The addressed database does not exist or could not be opened.
- db:no-index: the full-text index is not available.

**Examples**
Returns the number of occurrences for a single, specific index entry:
```
let $term := ft:tokenize($term)
return number(ft:tokens('db', $term)[. = $term]/@count)
```

### General Functions

### ft:contains

**Signatures**
- `ft:contains($input as item()*, $terms as item()*) as xs:boolean`, `ft:contains($input as item()*, $terms as item()*, $options as map(*)?) 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 exist:

- **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 if 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 | specified options are conflicting.

Examples | Checks if jack or john occurs in the input string John Doe:

```xml
ft:contains("John Doe", ("jack", "john"), map { "mode": "any" })
```

- Calls the function with stemming turned on and off:

```xml
(true(), false()) ! ft:contains("Häuser", "Haus", map { 'stemming': ., 'language': 'de' })
```

**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:tokenize**

Signatures | `ft:tokenize($string as xs:string?) as xs:string*, ft:tokenize($string as xs:string?, $options as map(*)) as xs:string`

Summary | Tokenizes the given $string, using the current default full-text options or the $options specified as second argument, and returns a sequence with the tokenized string. 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 if 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.

**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($string as xs:string?) as xs:string,
ft:normalize($string as xs:string?, $options as map(*)?) as xs:string
```

**Summary**

Normalizes the given $string, using the current default full-text options or the $options specified as second argument. The function accepts the same arguments as ft:tokenize; special characters and separators will be preserved.

**Examples**
- ft:normalize("Häuser am Meer", map { 'case': 'sensitive' }) returns the string Hauser am Meer.

**ft:thesaurus**

**Signatures**

```
ft:thesaurus($node as node(), $term as xs:string) as xs:string*,
ft:thesaurus($node as node(), $term as xs:string, $options as map(*)?) as xs:string*
```

**Summary**

Looks up a $term in a Thesaurus Structure supplied by $node. The following $options exist:
- relationship: determines the relationship between terms
- levels: determines the maximum number of levels to traverse

**Examples**

Returns happy and lucky:

```
ft:thesaurus(
  <thesaurus>
    <entry>
      <term>happy</term>
      <synonym>
        <term>lucky</term>
        <relationship>RT</relationship>
      </synonym>
    </entry>
  </thesaurus>,
  'happy'
)
```

**Highlighting Functions**

**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 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 supplied node must be a Database Node. As shown in Example 3, `update` or `transform` can be utilized to convert a fragment to the required internal representation.

**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":

```xml
ft:mark(db:get('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:

```xml
let $start := 1
let $end := 10
let $term := 'welcome'
for $ft in (db:get('DB')//*[text() contains text {$term}])[position() = $start to $end]
return element hit {
    ft:mark($ft[text() contains text {$term}])
}
```

**Example 3:** The following expression returns `<xml>hello <b>word</b></xml>`:

```xml
copy $p := <xml>hello world</xml>
modify ()
return ft:mark($p[text() contains text 'word'], 'b')
```

**ft:extract**

**Signatures**

```xml
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 `$nodes` of a full-text index request and chops irrelevant sections of the result. The default element name of the marker element is `mark`. An alternative element name can be chosen via the optional `$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 readability 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":

```xml
ft:extract(db:get('DB')//*[text() contains text 'hello'], 'b', 1)
```

**Errors**

**Code**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
</tr>
</tbody>
</table>

**Changelog**

Version 9.6

• Added: `ft:thesaurus`

• Updated: `ft:search`, `ft:contains`: new errors option.
Version 9.1
• Updated: `ft:tokenize` and `ft:normalize` can be called with empty sequence.

Version 9.0
• Updated: error codes updated; errors now use the module namespace

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 46. Hashing Module

This XQuery Module provides functions that perform different hash operations.

Conventions

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

Functions

### hash:md5

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hash:md5($value as xs:anyAtomicType) as xs:base64Binary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Computes the MD5 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.</td>
</tr>
<tr>
<td>Examples</td>
<td>• string(xs:hexBinary(hash:md5(&quot;BaseX&quot;))) returns 0D65185C9E296311C0A2200179E479A2.</td>
</tr>
<tr>
<td></td>
<td>• string(hash:md5(xs:base64Binary(&quot;&quot;))) returns 1B2M2Y8AsqTpgAmY7PhCfg=.</td>
</tr>
</tbody>
</table>

### hash:sha1

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hash:sha1($value as xs:anyAtomicType) as xs:base64Binary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Computes the SHA-1 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.</td>
</tr>
<tr>
<td>Examples</td>
<td>• string(xs:hexBinary(hash:sha1(&quot;BaseX&quot;))) returns 3AD5958F0E27D5AFFDCA2957560F121D0597A4ED.</td>
</tr>
<tr>
<td></td>
<td>• string(hash:sha1(xs:base64Binary(&quot;&quot;))) returns 2jmj715rSw0yVb/vlWAYkK/Yiiwk=.</td>
</tr>
</tbody>
</table>

### hash:sha256

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hash:sha256($value as xs:anyAtomicType) as xs:base64Binary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Computes the SHA-256 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary.</td>
</tr>
<tr>
<td>Examples</td>
<td>• string(xs:hexBinary(hash:sha256(&quot;BaseX&quot;))) returns 15D570763DE875D728BB69643392873B835C0CC94A2F1E881909DA47662821A3.</td>
</tr>
<tr>
<td></td>
<td>• string(hash:sha256(xs:base64Binary(&quot;&quot;))) returns 47DEQpj8HBSa+/TImW+5JceuQeRkm5NmpJW2G3hSuFU=.</td>
</tr>
</tbody>
</table>

### hash:hash

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hash:hash($value as xs:anyAtomicType, $algorithm as xs:string) as xs:base64Binary.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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 algorithms are supported: MD5, SHA-1, and SHA-256.</td>
</tr>
</tbody>
</table>
Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm</td>
<td>The specified hash algorithm is unknown.</td>
</tr>
</tbody>
</table>

Examples

- `string(xs:hexBinary(hash:hash("", "MD5")))` returns D41D8CD98F00B204E9800998ECF8427E.
- `string(hash:hash("", ""))` raises an error, because no algorithm was specified.

Errors

Changelog

Version 9.0
- Updated: error codes updated; errors now use the module namespace

The module was introduced with Version 7.3.
Chapter 47. 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.

Loops

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 does not 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 predicate function $pred to $start. If the result is false, $f is invoked with the start value – or, subsequently, with the result of this function – until the predicate function returns true().

Examples

- Doubles a numeric value until a maximum is reached:

  hof:until(
    function($output) { $output ge 1000 },
    function($input) { 2 * $input },
    1
  )

- Calculates the square root of a number by iteratively improving an initial guess:

  let $sqrt := function($input as xs:double) as xs:double {
    hof:until(
      function($result) { abs($result * $result - $input) < 0.00001 },
      function($guess) { ($guess + $input div $guess) div 2 },
      $input
    )
  )
  return $sqrt(25)

- Returns OK, as the predicate is evaluated first:

  hof:until(
    function($_) { true() },
    function($_) { error() },
  )
Higher-Order Functions Module

**hof:scan-left**

**Signatures**

\[
\text{hof:scan-left($seq$ as item()*, $start$ as item()*, $f$ as function(item()*, item()) as item()*) as item()*}
\]

**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:

```xml
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:

  ```xml
  hof:scan-left(1 to 10, 0, function($a, $b) { $a + $b })
  ```

**hof:take-while**

**Signatures**

\[
\text{hof:take-while($seq$ as item()*, $pred$ as function(item()) as xs:boolean) as item()*}
\]

**Summary**

The function returns items of $seq$ as long as the predicate $pred$ is satisfied. It is equivalent to:

```xml
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:

  ```xml
  hof:take-while(
      (1 to 100) ! random:integer(50),
      function($x) { $x >= 10 })
  ```

**hof:drop-while**

**Signatures**

\[
\text{hof:drop-while($seq$ as item()*, $pred$ as function(item()) as xs:boolean) as item()*}
\]

**Summary**

The function skips all items of $seq$ until the predicate $pred$ is not satisfied anymore. It is equivalent to:

```xml
declare function hof:drop-while($seq, $pred) {
    if($pred(head($seq))) then ( hof:drop-while(tail($seq), $pred)
    ) else {
        $seq
    }
};
```

**Examples**

Returns the name of the first file that does not exist on disk:

```xml
hof:drop-while(
    (1 to 1000) ! (. || '.log'),
    file:exists?#1
)[1]
```
# Higher-Order Functions Module

## Sorting

**hof:top-k-by**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hof:top-k-by($seq as item()<em>, $sort-key as function(item()) as item(), $k as xs:integer) as item()</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>(for $x$ in $seq$ order by $sort-key($x) descending return $x$)[position() &lt;= $k]</td>
</tr>
<tr>
<td>Examples</td>
<td>• hof:top-k-by(1 to 1000, hof:id#1, 5) returns 1000 999 998 997 996</td>
</tr>
<tr>
<td></td>
<td>• hof:top-k-by(1 to 1000, function($x) { -$x }, 3) returns 1 2 3</td>
</tr>
<tr>
<td></td>
<td>• hof:top-k-by(&lt;x a='1' b='2' c='3'/&gt;/@*, xs:integer#1, 2)/node-name() returns c b</td>
</tr>
</tbody>
</table>

**hof:top-k-with**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hof:top-k-with($seq as item()<em>, $lt as function(item(), item()) as xs:boolean, $k as xs:integer) as item()</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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, $k).</td>
</tr>
<tr>
<td>Examples</td>
<td>• hof:top-k-with(1 to 1000, function($a, $b) { $a lt $b }, 5) returns 1000 999 998 997 996</td>
</tr>
<tr>
<td></td>
<td>• hof:top-k-with(-5 to 5, function($a, $b) { abs($a) gt abs($b) }, 5) returns 0 1 -1 2 -2</td>
</tr>
</tbody>
</table>

## IDs

**hof:id**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>hof:id($expr as item()<em>) as item()</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns its argument unchanged. This function isn’t useful on its own, but can be used as argument to other higher-order functions.</td>
</tr>
<tr>
<td>Examples</td>
<td>• hof:id(1 to 5) returns 1 2 3 4 5</td>
</tr>
<tr>
<td></td>
<td>• With higher-order functions:</td>
</tr>
<tr>
<td></td>
<td>let $sort := sort(? , (), hof:id#1)</td>
</tr>
<tr>
<td></td>
<td>let $reverse-sort := sort(? , (), function($x) { -$x })</td>
</tr>
<tr>
<td></td>
<td>return (</td>
</tr>
<tr>
<td></td>
<td>$sort((1, 5, 3, 2, 4)),</td>
</tr>
<tr>
<td></td>
<td>'</td>
</tr>
<tr>
<td></td>
<td>$reverse-sort((1, 5, 3, 2, 4))</td>
</tr>
<tr>
<td></td>
<td>)</td>
</tr>
<tr>
<td></td>
<td>returns: 1 2 3 4 5</td>
</tr>
</tbody>
</table>

**hof:const**

| Signatures | hof:const($expr as item()*, $ignored as item()*) as item()* |

272
**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:

```plaintext
let $zip-sum := function($f, $seq1, $seq2) {
    sum(for-each-pair($seq1, $seq2, $f))
}
let $sum-all := $zip-sum(function($a, $b) { $a + $b }, ?, ?)
let $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.

```plaintext
let $insert-with := function($f, $map, $k, $v) {
    let $old := $map($k)
    let $new := if($old) then $f($v, $old) else $v
    return map:merge(($map, map:entry($k, $new)))
}
let $map := map { 'foo': 1 }
let $add := $insert-with(function($a, $b) { $a + $b }, ?, ?, ?)
let $ins := $insert-with(hof:const#2, ?, ?, ?)
return ($add($map, 'foo', 2)('foo'),
        $ins($map, 'foo', 42)('foo'))
returns 3 42
```

**Changelog**

Version 9.5

- Added: `hof:drop-while`

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 48. HTML Module

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 and errors in this module are assigned to the http://basex.org/modules/html namespace, which is statically bound to the html prefix.

Functions

html:doc

Signatures

html:doc($uri as xs:string?) as document-node()?, html:doc($uri as xs:string?, $options as map(*)?) as document-node()?,

Summary

Fetches the HTML document referred to by the given $uri, converts it to XML and returns a document node. The $options argument can be used to set TagSoup Options.

Errors

parse: the input cannot be converted to XML.

html:parse

Signatures

html:parse($input as xs:anyAtomicType) as document-node(), html:parse($input as xs:anyAtomicType, $options as map(*)?) as document-node(),

Summary

Converts the HTML document specified by $input to XML and returns a document node:

• The input may be of type xs:string, xs:base64Binary, or xs:hexBinary.

• If the input is passed on in its binary representation, the HTML parser will try to choose the correct encoding automatically.

The $options argument can be used to set TagSoup Options.

Errors

parse: the input cannot be converted to XML.

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.

Examples

Basic Example

The following query converts the specified string to an XML document node.

Query

html:parse("<html>")

Result

274
Specifying Options

The next query creates an XML document with namespaces:

Query

```html
html:parse("<a href='ok.html'/>", map { 'nons': false() })
```

Result

```html
<html xmlns="http://www.w3.org/1999/xhtml">
  <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
```

Result

```html
<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"/>
    ...
</head>
<body>
</html>
```

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse</td>
<td>The input cannot be converted to XML.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.4

- Added: html:doc

Version 9.0

- Updated: error codes updated; errors now use the module namespace

The module was introduced with Version 7.6.
Chapter 49. HTTP Client Module

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.

If `<http:header name="Accept-Encoding" value="gzip"/>` is specified and if the addressed web server provides support for the gzip compression algorithm, the response will automatically be decompressed.

**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 `experr` prefix.

**Functions**

**http:send-request**

**Signatures**

- `http:send-request($request as element(http:request)) as item()`,
- `http:send-request($request as element(http:request)?, $href as xs:string?) as item()`,
- `http:send-request($request as element(http:request)?, $href as xs:string?, $bodies as item()* 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 `$request` is used instead.
- The request body can also be supplied via the `$bodies` parameter.
- Certificate verification can be globally disabled via the `IGNORECERT` option.

**Notes:**

- Both basic and digest authentication is supported.
- While the contents of the request can be supplied as child of the `http:body` element, it is faster and safer to pass them on via the third argument.
- For further information, please check out the EXPath specification.

**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.
HTTP Client Module

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="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 the Google search home page with a timeout of 10 seconds. In order to parse HTML, TagSoup must be contained in the class path.

Query:

http:send-request(http:request method='get' href='http://www.google.com' timeout='10'/>)

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>

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 Client Module

```php
<html:request name="User-Agent" value="Opera" />
try {
    html:parse($binary)
} catch * {
    'Conversion to XML failed: ' || $err:description
}
```

### SVG Data

Content-type ending with +xml, e.g. image/svg+xml.

**Query:**

```php
http:send-request(<http:request method='get'/>, 'http://upload.wikimedia.org/wikipedia/commons/6/6b/Bitmap_VS_SVG.svg')
```

**Result:**

```xml
<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>
    ...
  </defs>
</svg>
```

### POST Request

POST request to the BaseX REST Service, specifying a username and password.

**Query:**

```php
http:send-request(
    <http:request method='post' username='admin' password='admin'>
        <http:body media-type='application/xml'/>
    </http:request>,
    'http://localhost:8984/rest',
    <query>
        <text>
            for $i in 1 to 3
                return <div>Section {$i }</div>
        </text>
    </query>
)
```
HTTP Client Module

Result:

```xml
  <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>
```

File Upload

Performs an HTML file upload. In the RESTXQ code, the uploaded file is written to the temporary directory:

Query:

```xml
let $path := 'file-to-be.uploaded'
return http:send-request(
  <http:request method='POST'>
    <http:multipart media-type='multipart/form-data'>
      <http:header name='content-disposition' value='form-data; name="files"; filename="{ file:name($path) }"'/>
      <http:body media-type='application/octet-stream'/>
    </http:multipart>
  </http:request>,
  'http://localhost:8984/write-to-temp',
  file:read-binary($path)
)
```

RESTXQ service:

```xml
declare
%rest:POST
%rest:path('/write-to-temp')
%rest:form-param('files', '{$files}')
function dba:file-upload(
  $files  as map(xs:string, xs:base64Binary)
) as empty-sequence() {
  map:for-each($files, function($file, $content) {
    file:write-binary(file:temp-dir() || $file, $content)
  });
}
```

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HC0001</td>
<td>An HTTP error occurred.</td>
</tr>
<tr>
<td>HC0002</td>
<td>Error parsing the entity content as XML or HTML.</td>
</tr>
<tr>
<td>HC0003</td>
<td>With a multipart response, the override-media-type must be either a multipart media type or application/octet-stream.</td>
</tr>
<tr>
<td>HC0004</td>
<td>The src attribute on the body element is mutually exclusive with all other attribute (except the media-type).</td>
</tr>
<tr>
<td>HC0005</td>
<td>The request element is not valid.</td>
</tr>
<tr>
<td>HC0006</td>
<td>A timeout occurred waiting for the response.</td>
</tr>
</tbody>
</table>
**Changelog**

Version 9.0

• Updated: support for gzipped content encoding

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 50. Index Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for displaying information stored in the database index structures.

For functions that use the indexes to return nodes see Value Indexes in the Database Module and ft:search in the Full-Text Module.

Conventions

All functions and errors in this module are assigned to the http://basex.org/modules/index namespace, which is statically bound to the index 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

db:get: 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

db:get: The addressed database does not exist or could not be opened.db:no-index: the 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.
Index Module

Errors

| db:get: The addressed database does not exist or could not be opened. |
| db:no-index: the index is not available. |

index:tokens

Signatures

| index:tokens($db as xs:string) as element(value)* |

Summary

Returns all strings stored in the Token Index of the database $db, along with their number of occurrences.

Errors

| db:get: The addressed database does not exist or could not be opened. |
| db:no-index: the 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

| db:get: 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

| db:get: The addressed database does not exist or could not be opened. |

Changelog

Version 8.4
• Added: index:tokens

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 51. 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 and errors 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.

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 specified resource will be retrieved as string and compiled, and its functions will be added to the query context and returned to the user. A relative URI will be resolved against the static base URI of the query.

Examples

Invokes the declared functions and returns their 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()
```

Errors

parse: Error while parsing a module.

inspect:function-annotations

Signatures

inspect:function-annotations($function as function(*)?) as map(xs:QName, xs:anyAtomicType*).

Summary

Returns the annotations of the specified $function in a map.

Examples

```
• Returns an empty map:
  inspect:function-annotations(true#0)
• Returns a map with a single key Q{http://www.w3.org/2012/xquery}private and an empty sequence as value:
  declare %private function local:f() { 'well hidden' };
  inspect:function-annotations(local:f#0)
```

inspect:static-context

Signatures

inspect:static-context($function as function(*)?, $name as xs:string) as item(*)

283
### Summary

Returns a component of the static context of a $function with the specified $name. If no function is supplied, the current static context is considered. The following components can be requested:

- **base-uri**: Static base URI.
- **namespaces**: Prefix/URI map with all statically known namespaces.
- **element-namespace**: Default element/type namespace URI, or an empty sequence if it is absent.
- **function-namespace**: Default function namespace URI, or an empty sequence if it is absent.
- **collation**: URI of the default collation.
- **ordering**: Ordering mode (ordered/unordered)
- **construction**: Construction mode (preserve/strip)
- **default-order-empty**: Default order for empty sequences (greatest/least)
- **boundary-space**: Boundary-space policy (preserve/strip)
- **copy-namespaces**: Copy-namespaces mode (inherit/no-inherit,preserve/no-preserve)
- **decimal-formats**: Nested map with all statically known decimal formats

### Examples

- Returns the static base URI (same as static-base-uri()):
  
  ```xml
  inspect:static-context((), 'base-uri')
  ```

- Returns a map with all namespaces that are statically known in the module of the specified function:
  
  ```xml
  import module namespace data = 'data.xqm';
  inspect:static-context(data:get#1, 'namespaces')
  ```

### Errors

**unknown**: The specified component does not exist.

### Documentation

**inspect:type**

#### Signatures

inspect:type($value as item()*) as xs:string, inspect:type($value as item()*, $options as map(*)) as xs:string

#### Summary

Returns a string representation of the type of a $value:

- The string includes the occurrence indicator.
- The type of functions and nodes may be stricter than the returned type.
- For type checking, the standard expressions typeswitch and instance of should be used instead.

The following $options are available:

- **item**: If enabled, only the item type is returned and the occurrence indicator is omitted. The default is false().
### inspection Module

- **mode**: If value is specified, the assigned type of the result value is returned. With expression the type of the input expression is returned (please note that the original expression may already have been rewritten at compile-time). With computed, the exact value is computed at runtime, based on the expression and the result value. The default is computed.

#### Examples

- `inspect:type((<a/>, <a/>))` yields `element(a)+`
- `inspect:type(map { 'a': (1, 2)[. = 1] })` yields `map(xs:string, xs:integer)`
- `inspect:type(1 to 100, map { 'item': true() })` yields `xs:integer`

### 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:

```xml
<function name="count" uri="http://www.w3.org/2005/xpath-functions" external="false">
  <argument type="item()" occurrence="*"/>
  <return type="xs:integer"/>
</function>
```

The function...

```xml
(:~
  : 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...

```xml
<function name="local:same" uri="http://www.w3.org/2005/xquery-local-functions" external="false">
  <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:

```xml
inspect:context()/function ! function-lookup(QName($uri, @name), 0) ! . ()
```

Return the names of all private functions in the current context:

```xml
for $f in inspect:context()/function
where $f/annotation/@name = 'private'
```
**inspect:module**

**Signatures**
inspect:module($uri as xs:string) as element(module)

**Summary**
Retrieves the resource located at the specified $uri, parses it as XQuery module, and generates an element that describes the module’s structure. A relative URI will be resolved against the static base URI of the query.

**Examples**
An example is shown below.

**Errors**
parse: Error while parsing a module.

---

**inspect:xqdoc**

**Signatures**
inspect:xqdoc($uri as xs:string) as element(xqdoc:xqdoc)

**Summary**
Retrieves the resource located at the specified $uri, parses it as XQuery module, and generates an xqDoc element. A relative URI will be resolved against the static base URI of the query. xqDoc provides a simple vendor-neutral solution for generating 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 more recent versions 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.

**Examples**
An example is shown below.

**Errors**
parse: Error while parsing a module.

---

**Examples**

This is the sample.xqm library module:

```xml
<-- 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. -->
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:

```xml
<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" external="false">
    <description>This is a sample string.</description>
  </variable>
  <function name="samples:same" uri="http://basex.org/modules/samples" external="false">
    <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:

```xml
<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: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:comment>
      <xqdoc:param>$number number to return</xqdoc:param>
      <xqdoc:return>specified number</xqdoc:return>
    </xqdoc:function>
    <xqdoc:namespace>samples:same</xqdoc:namespace>
    <xqdoc:annotations>
      <xqdoc:annotation name="private"/>
    </xqdoc:annotations>
  </xqdoc:functions>
</xqdoc:xqdoc>
```
Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parse</td>
<td>Error while parsing a module.</td>
</tr>
<tr>
<td>unknown</td>
<td>The specified component does not exist.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.6

- Updated: inspect:type: options added

Version 9.3

- Added: inspect:type

Version 8.5

- Added: inspect:function-annotations, inspect:static-context
- Updated: external attribute added to variables and functions
- Updated: Relative URIs will always be resolved against the static base URI of the query

Version 7.9

- Updated: a query URI can now be specified with inspect:functions.

This module was introduced with Version 7.7.
Chapter 52. Job Module

Read this entry online in the BaseX Wiki.

Updated with Version 10: Renamed from Jobs Module to Job Module. The namespace URI has been updated as well.

This XQuery Module provides functions for organizing scheduled, queued, running and cached jobs. Jobs can be commands, queries, client, or HTTP requests.

Conventions

All functions in this module are assigned to the http://basex.org/modules/job namespace, which is statically bound to the job prefix. Errors will be bound to the same prefix.

Services

A job can be registered as service by supplying the service option to job:eval:

(: register job as service; will be run every day at 1 am :) 
job:eval('db:drop("tmp")', (), map { 'id':'cleanup', 'start':'01:00:00', 'interval':'P1D', 'service': true() }),

(: list registered services :) 
job:services(),

(: result: <job base-uri="..." id="cleanup" interval="P1D" start="01:00:00">db:drop("tmp")</job> :) 

(: unregister job :) 
job:remove('cleanup', map { 'service': true() })

Some more notes:

• All job services will be scheduled for evaluation when the BaseX server or BaseX HTTP server is started.

• If a job service is outdated (e.g. because a supplied end time has been exceeded), it will be removed from the jobs file at startup time.

• The job definitions are stored in a jobs.xml file in the database directory. It can also be edited manually.

Executing Jobs

There are cases in which a client does not, or cannot, wait until a request is fully processed. The client may be a browser, which sends an HTTP request to the server to start another time-consuming query job. The functions in this section allow you to register a new query job from a running query. Jobs can be executed immediately (i.e., as soon as the Concurrency Control allows it) or scheduled for repeated execution. Each registered job gets a job ID, and the ID can be used to retrieve a query result, stop a job, or wait for its termination.

job:eval

Signatures

job:eval($query as xs:anyAtomicType) as xs:string, 
job:eval($query as xs:anyAtomicType, $bindings as map(*)?) as xs:string, 
job:eval($query as xs:anyAtomicType, $bindings as map(*)?, $options as map(*)?) as xs:string.

Summary

Schedules the evaluation of the supplied $query(xs:string, or of type xs:anyURI, pointing to a resource), and returns a query ID. The query will be queued, and the result will optionally be cached. Queries can be updating. Variables and the context value can be declared via $bindings (see xquery:eval for more details). The following $options can be supplied:

• cache: indicates if the query result will be cached or ignored (default: false):
The result will be cached in main-memory until it is fetched via \texttt{job:result}, or until \texttt{CACHETIMEOUT} is exceeded.

If the query raises an error, it will be cached and returned instead.

\texttt{start}: a \texttt{dayTimeDuration}, \texttt{time}, \texttt{dateTime} or integer can be specified to delay the execution of the query:

- If a \texttt{dayTimeDuration} is specified, the query will be queued after the specified duration has passed. Examples of valid values are: \texttt{P1D} (1 day), \texttt{PT5M} (5 minutes), \texttt{PT0.1S} (100 ms). An error will be raised if a negative value is specified.

- If a \texttt{dateTime} is specified, the query will be executed at this date. Examples for valid values are: \texttt{2018-12-31T23:59:59} (New Year's Eve 2018, close to midnight). An error will be raised if the specified time lies in the past.

- If a time is specified, the query will be executed at this time of the day. Examples of valid times are: \texttt{02:00:00} (2am local time), \texttt{12:00:00Z} (noon, UTC). If the time lies in the past, the query will be executed the next day.

- An integer will be interpreted as minutes. If the specified number is greater than the elapsed minutes of the current hour, the query will be executed one hour later.

\texttt{interval}: a \texttt{dayTimeDuration} string can be specified to execute the query periodically. An error is raised if the specified interval is less than one second (\texttt{PT1S}). If the next scheduled call is due, and if a query with the same ID is still running, it will be skipped.

\texttt{end}: scheduling can be stopped after a given time or duration. The string format is the same as for \texttt{start}. An error is raised if the resulting end time is smaller than the start time.

\texttt{base-uri}: sets the \texttt{base-uri} property for the query. This URI will be used when resolving relative URIs, such as with \texttt{fn:doc}.

\texttt{id}: sets a custom job ID. The ID must not start with the standard \texttt{job} prefix, and it can only be assigned if no job with the same name exists.

\texttt{service}: additionally registers the job as \texttt{service}. Registered services must have no variable bindings.

\texttt{log}: writes the specified string to the database logs. Two log entries are stored, one at the beginning and another one after the execution of the job.

\textbf{Errors}

\texttt{overflow}: Query execution is rejected because too many jobs are queued or being executed. \texttt{CACHETIMEOUT} can be decreased if the default setting is too restrictive.

\texttt{range}: A specified time or duration is out of range.

\texttt{id}: The specified ID is invalid or has already been assigned.

\texttt{options}: The specified options are conflicting.

\textbf{Examples}

- Cache query result. The returned ID can be used to pick up the result with \texttt{job:result}:

  \begin{verbatim}
  job:eval("1+3", (), map { 'cache': true() })
  \end{verbatim}

- A happy birthday mail will be sent at the given date:

  \begin{verbatim}
  job:eval("import module namespace mail='mail'; mail:send('Happy birthday!')", (), map { 'start': '2018-09-01T06:00:00' })}
  \end{verbatim}

- The following \texttt{RESTXQ} functions can be called to execute a query at 2 am every day. An ID will be returned by the first function, which can be used to stop the scheduler via the second function:

  \begin{verbatim}
  declare %rest:POST("${query}")) %rest:path('/start-scheduling') function
  local:start($query) {
  \end{verbatim}
• Query execution is scheduled for every second, and for 10 seconds in total. As the query itself will take 1.5 seconds, it will only be executed every second time:

```xml
job:eval("prof:sleep(1500)", (), map { 'interval': 'PT1S', 'end': 'PT10S' })
```

• The query in the specified file will be evaluated once:

```xml
job:eval(xs:anyURI('cleanup.xq'))
```

• The following expression, if stored in a file, will be evaluated every 5 seconds:

```xml
job:eval(
    static-base-uri(),
    map { },
    map { 'start': 'PT5S' } )
```

### job:result

**Version 10:** options argument added.

**Signatures**

```xml
job:result($id as xs:string) as item()*, job:result($id as xs:string, $options as map(*)) as item()*
```

**Summary**

Returns the cached result of a job with the specified job $id:

• If the original job has raised an error, the cached error will be raised instead.

• The cached result or error will be dropped after it has been retrieved.

• If the result has not been cached or if it has been dropped, an empty sequence is returned.

The following $options can be supplied:

• **keep**: Keep the cached result or error after retrieval.

**Examples**

• The following RESTXQ function will either return the result of a previously started job or raise an error:

```xml
declare %rest:path('/result/{$id}') function local:result($id) {
    job:result($id)
};
```

• The following query demonstrates how the results of an executed query can be returned within the same query (see below why you should avoid this pattern in practice):

```xml
let $query := job:eval('(1 to 1000000)[. = 1]', map { }, map { 'cache': true() } )
return (
    job:wait($query),
    job:result($query)
)
```

Queries of this kind can cause deadlocks! If the original query and the new query perform updates on the same database, the second query will only be run after the first one has been executed, and the first query will wait for the second query forever. You should resort to xquery:fork-join if you want to have full control on parallel query execution.
**job:remove**

*Updated with Version 10:* Renamed from `jobs:stop`.

**Signatures**

```xml
job:remove($id as xs:string) as empty-sequence().
job:remove($id as xs:string, $options as map(*)?) as empty-sequence()
```

**Summary**

Triggers the cancelation of a job with the specified `$id`, cancels a scheduled job or removes a cached result. Unknown IDs are ignored. All jobs are gracefully stopped; it is up to the process to decide when it is safe to shut down. The following `$options` can be supplied:

- **service**: additionally removes the job from the job services list.

**Examples**

- `job:list()[. != job:current()] ! job:remove(.)` stops and discards all jobs except for the current one.
- `job:remove(job:current())` interrupts the current job.

---

**job:wait**

**Signatures**

```xml
job:wait($id as xs:string) as empty-sequence()
```

**Summary**

Waits for the completion of a job with the specified `$id`:

- The function will terminate immediately if the job ID is unknown. This is the case if a future job has not been queued yet, or if the ID has already been discarded after job evaluation.
- If the function is called with the ID of a queued job, or repeatedly executed job, it may stall and never terminate.

**Errors**

self: The current job is addressed.

---

**Listing Jobs**

**job:current**

**Signatures**

```xml
job:current() as xs:string
```

**Summary**

Returns the ID of the current job.

**job:list**

**Signatures**

```xml
job:list() as xs:string*
```

**Summary**

Returns the IDs of all jobs that are currently registered. The list includes scheduled, queued, running, stopped, and finished jobs with cached results.

**Examples**

`job:list()` returns the same job ID as `job:current` if no other job is registered.

**job:list-details**

**Signatures**

```xml
job:list-details() as element(job)*.
job:list-details($id as xs:string) as element(job)*
```

**Summary**

Returns information on all jobs that are currently registered, or on a job with the specified `$id` (or an empty sequence if this job is not found). The list includes scheduled, queued, running jobs, and cached jobs. A string representation of the job, or its URI, will be returned as a value. The returned elements have additional attributes:

- **id**: job ID
- **type**: type of the job (command, query, REST, RESTXQ, etc.)
- **state**: current state of the job: scheduled, queued, running, cached
Job Module

- **user**: user who started the job
- **duration**: evaluation time (included if a job is running or if the result was cached)
- **start**: next start of job (included if a job will be executed repeatedly)
- **time**: time when job was registered

**Examples**

`job:list-details()` returns information on the currently running job and possibly others:

```xml
<job id="job1" type="XQuery" state="running" user="admin"
     duration="PT0.001S">
  XQUERY job:list-details()
</job>
```

### job:bindings

*Introduced with Version 10.*

**Signatures**

`job:bindings($id as xs:string) as map(*)`

**Summary**

Returns the variable bindings of an existing job with the specified `$id`. If no variables have been bound to this job, an empty map is returned.

### job:finished

**Signatures**

`job:finished($id as xs:string) as xs:boolean`

**Summary**

Indicates if the evaluation of an already running job with the specified `$id` has finished. As the IDs of finished jobs will usually be discarded, unless caching is enabled, the function will also return `true` for unknown jobs.

- `false` indicates that the job ID is scheduled, queued, or currently running.
- `true` will be returned if the job has either finished, or if the ID is unknown (because the IDs of all finished jobs will not be cached).

### job:services

**Signatures**

`job:services() as element(job)*`

**Summary**

Returns a list of all jobs that have been persistently registered as Services.

**Errors**

services: Registered services cannot be parsed.

### Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>The specified options are conflicting.</td>
</tr>
<tr>
<td>id</td>
<td>The specified ID is invalid or has already been assigned.</td>
</tr>
<tr>
<td>overflow</td>
<td>Too many queries or query results are queued.</td>
</tr>
<tr>
<td>range</td>
<td>A specified time or duration is out of range.</td>
</tr>
<tr>
<td>running</td>
<td>A query is still running.</td>
</tr>
<tr>
<td>self</td>
<td>The current job cannot be addressed.</td>
</tr>
<tr>
<td>service</td>
<td>Registered services cannot be parsed, added or removed.</td>
</tr>
</tbody>
</table>

### Changelog

Version 10.0
• Updated: Renamed from Jobs Module to Job Module. The namespace URI has been updated as well.

• Updated: job:remove renamed from jobs:stop.

• Updated: job:result: options argument added.

• Added: job:bindings

Version 9.7

• Updated: job:result: return empty sequence if no result is cached.

Version 9.5

• Updated: job:eval: integers added as valid start and end times.

Version 9.4

• Updated: job:eval: option added for writing log entries.

• Updated: job:list-details: interval added.

Version 9.2

• Deleted: job:invoke (merged with job:eval)

Version 9.1

• Updated: job:list-details: registration time added.

Version 9.0

• Added: job:invoke, Services

Version 8.6

• Updated: job:eval: id option added.

The module was introduced with Version 8.5.
Chapter 53. JSON Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to parse and serialize JSON data. JSON (JavaScript Object Notation) is a popular data exchange format for applications written in JavaScript. As there are notable differences between JSON and XML, or XQuery data types, 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 and errors in this module are assigned to the http://basex.org/modules/json namespace, which is statically bound to the json prefix.

Conversion Formats

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.

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 encoded, as described in the Conversion Module, and used as element name.
• Array entries are also represented via elements, with _ 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.

Basic

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.

XQuery

The xquery format is lossless, too. It converts JSON data to an XQuery value (a map, array, string, number, boolean, or empty sequence) and vice versa. The conversion rules are the same as for fn:parse-json.

The resulting representation consumes less memory than XML-based formats, and values can be directly accessed without conversion. Thus, it is recommendable for very large inputs and for efficient ad-hoc processing.

Options

The following options are available (the Direction column indicates if an option applies to parsing, serialization, or both operations):

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>Specifies the format for converting JSON data.</td>
<td>direct, attributes, jsonml, xquery</td>
<td>direct</td>
<td>parse, serialize</td>
</tr>
<tr>
<td>liberal</td>
<td>Determines if minor deviations from RFC 7159 will be ignored.</td>
<td>yes, no</td>
<td>no</td>
<td>parse</td>
</tr>
<tr>
<td>merge</td>
<td>This option is considered when direct or attributes conversion is used:</td>
<td>yes, no</td>
<td>no</td>
<td>parse, serialize</td>
</tr>
<tr>
<td></td>
<td>• If a name has the same type throughout the data, the type attribute will be omitted. Instead, the name will be listed in additional, type-specific attributes in the root node.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strings</td>
<td>Indicates if type attributes will be added for strings.</td>
<td>yes, no</td>
<td>yes</td>
<td>parse, serialize</td>
</tr>
<tr>
<td>lax</td>
<td>Specifies if a lax approach is used to convert QNames to JSON names.</td>
<td>yes, no</td>
<td>no</td>
<td>parse, serialize</td>
</tr>
<tr>
<td>escape</td>
<td>Indicates if escaped characters are expanded (for example, \n becomes a single x0A character, while \u20AC becomes the character €).</td>
<td>yes, no</td>
<td>yes</td>
<td>parse</td>
</tr>
<tr>
<td>escape</td>
<td>Indicates if characters are escaped whenever the JSON syntax requires it. This option can be set to no if strings are already in escaped form and no further escaping is permitted.</td>
<td>yes, no</td>
<td>yes</td>
<td>serialize</td>
</tr>
<tr>
<td>indent</td>
<td>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 indentation serialization parameter.</td>
<td>yes, no</td>
<td>yes</td>
<td>serialize</td>
</tr>
</tbody>
</table>
Functions

**json:doc**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>json:doc($uri as xs:string) as item()?, json:doc($uri as xs:string, $options as map(*)?) as item()?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Fetches the JSON document referred to by the given $uri and converts it to an XQuery value. The $options argument can be used to control the way the input is converted.</td>
</tr>
<tr>
<td>Errors</td>
<td>parse: the specified input cannot be parsed as JSON document. options: the specified options are conflicting.</td>
</tr>
</tbody>
</table>

**json:parse**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>json:parse($string as xs:string?) as item()?, json:parse($string as xs:string?, $options as map(*)?) as item()?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Converts the JSON $string to an XQuery value. 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.</td>
</tr>
<tr>
<td>Errors</td>
<td>parse: the specified input cannot be parsed as JSON document. options: the specified options are conflicting.</td>
</tr>
</tbody>
</table>

**json:serialize**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>json:serialize($input as item()? as xs:string, json:serialize($input as item()?, $options as map(*)?) as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Serializes the specified $input as JSON, using the specified $options, and returns the result as string:</td>
</tr>
<tr>
<td></td>
<td>• The input is expected to conform to the results that are created by json:parse.</td>
</tr>
<tr>
<td></td>
<td>• Non-conforming items will be serialized as specified in the json output method of the official recommendation.</td>
</tr>
<tr>
<td>Values can also be serialized as JSON with the standard Serialization feature of XQuery:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• The parameter method needs to be set to json, and</td>
</tr>
<tr>
<td></td>
<td>• the options presented in this article need to be assigned to the json parameter.</td>
</tr>
<tr>
<td>Errors</td>
<td>serialize: the specified node cannot be serialized as JSON document.</td>
</tr>
</tbody>
</table>

Examples

BaseX Format

**Example 1: Adds all JSON documents in a directory to a database**

Query:

```xml
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:
Example 3: Converts a JSON string with simple objects and arrays

Query:

```json
let $options := map { 'merge': true() } return 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:

```json
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>
  </phone>
  <type>mobile</type>
  <number type="number">1327724623</number>
</json>

JsonML Format

Example 1: Converts all XML documents in a database to the JsonML format and writes them to disk

Query:

```xml
for $doc in collection('json')
let $name := document-uri($doc)
let $json := json:serialize($doc, map { 'format': 'jsonml' })
return file:write($name, $json)
```

Example 2: Converts an XML document with elements and text

Query:

```xml
json:serialize(doc('flickr.xml'), map { 'format': 'jsonml' })
```

flickr.xml:

```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:

```json
["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 3: Converts a document with nested elements and attributes to JsonML

Query:

```xml
"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/"]```
`json:serialize(doc('input.xml'), map { 'format': 'jsonml' })`

**input.xml:**

```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:**

```json
["address", {"id":"1"},
  ["last_name", "Smith"],
  ["age", "25"],
  ["address", ["street", "21 2nd Street"],
   ["city", "New York"],
   ["code", "10021"]],
  ["phone", ["type":"home"],
   "212 555-1234"]]
```

**XQuery Format**

**Example 1: Converts a JSON string to XQuery**

**Query:**

```xml
let $input := '{
  "Title": "Drinks",
  "Author": [ "Jim Daniels", "Jack Beam" ]
}'
let $data := json:parse($input, map { 'format': 'xquery' })
return map:for-each($data, function($k, $v) {
  $k || ': ' || string-join($v, ', ')
})
```

**Result:**

Author: Jim Daniels, Jack Beam
Title: Drinks

**Example 2: Converts XQuery data to JSON**

**Query:**

```xml
for $item in (true(), 'ABC', array { 1 to 5 },
  map { "Key": "Value" })
return json:serialize($item,
```
map { 'format': 'xquery', 'indent': 'no' }

Result:

true
"ABC"
[1,2,3,4,5]
{"Key":"Value"}

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>The specified options are conflicting.</td>
</tr>
<tr>
<td>parse</td>
<td>The specified input cannot be parsed as JSON document.</td>
</tr>
<tr>
<td>serialize</td>
<td>The specified node cannot be serialized as JSON document.</td>
</tr>
</tbody>
</table>

**Changelog**

Version 9.4

- Added: json:doc

Version 9.1

- Updated: json:parse can be called with empty sequence.

Version 9.0

- Updated: map format renamed to xquery.
- Updated: error codes updated; errors now use the module namespace

Version 8.4

- Updated: unescape changed to escape.

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

- 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 54. Lazy Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for handling lazy items.

In contrast to standard XQuery items, a lazy item contains a reference to the actual data, and the data itself will only be retrieved if it is processed. Hence, possible errors will be postponed, and no memory will be occupied by a lazy item as long as its content has not been requested yet.

The following BaseX functions return lazy items:

- Lazy Base64 binaries:
  - fetch:binary
  - file:read-binary
  - db:get-binary

- Lazy strings:
  - fetch:text
  - file:read-text

Some functions are capable of consuming the contents of lazy items in a streamable fashion: data will not be cached, but instead passed on to another target (file, the calling expression, etc.). The following streaming functions are currently available:

- Archive Module (most functions)
- Conversion Module: convert:binary-to-string
- File Module: file:write-binary-text, file:write-text (if no encoding is specified)
- Database Module: db:put-binary
- Hashing Module (all functions)

The XQuery expression below serves as an example on how large files can be downloaded and written to a file with constant memory consumption:

```
```

If lazy items are serialized, they will be streamed as well.

Conventions

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

Functions

**lazy:cache**

| Signatures | lazy:cache($items as item()) as item()* | lazy:cache($items as item()*, $lazy as xs:boolean) as item()* |
Lazy Module

Summary
Caches the data of lazy $items in a sequence:

• data of lazy items will be retrieved and cached inside the item.
• non-lazy items, or lazy items with cached data, will simply be passed through.
• If $lazy is set to true(), caching will be deferred until the data is eventually requested.

Streaming will be disabled: Data will always be cached before a stream is returned.

Caching is advisable if an item will be processed more than once, or if the data may not be available anymore at a later stage.

Example
In the following example, a file will be deleted before its content is returned. To avoid a “file not found” error when serializing the result, the content must be cached:

```xml
let $file := 'data.txt'
let $text := lazy:cache(file:read-text($file))
return (file:delete($file), $text)
```

lazy:is-lazy

Signatures
```xml
lazy:is-lazy($item as item()) as xs:boolean
```

Summary
Checks whether the specified $item is lazy.

lazy:is-cached

Signatures
```xml
lazy:is-cached($item as item()) as xs:boolean
```

Summary
Checks whether the contents of the specified $item are cached. The function will always return true for non-lazy items.

Changelog

Version 9.1
• Updated: lazy:cache: $lazy argument added; support for sequences.

Version 9.0
• Updated: Renamed from Streaming Module to Lazy Module.
• Added: lazy:is-cached

Version 8.0
• Updated: stream:materialize extended to sequences.

This module was introduced with Version 7.7.
Chapter 55. Map Module

This XQuery Module contains functions for manipulating maps. Maps have been introduced with XQuery 3.1.

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:

```xml
declare variable $week := map {
};
```

**map:contains**

**Signatures**

map:contains($map as map(*), $key as xs:anyAtomicType) as xs:boolean

**Summary**

Returns true if the supplied $map 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. 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:

```xml
map:merge((
  map:entry("Sun", "Sunday"),
  map:entry("Mon", "Monday"),
  map:entry("Tue", "Tuesday"),
  map:entry("Wed", "Wednesday"),
  map:entry("Thu", "Thursday"),
  map:entry("Fri", "Friday"),
  map:entry("Sat", "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:find

Signatures
map:find($input as item()*, $key as xs:anyAtomicType) as array(*)

Summary
Returns all values of maps in the supplied $input with the specified $key. The found values will be returned in an array. Arbitrary input will be processed recursively as follows:

- In a sequence, each item will be processed in order.
- In an array, all array members will be processed as sequence.
- In a map, all entries whose keys match the specified key. Moreover, all values of the map will be processed as sequence.

Examples
• map:find(map { 1:2 }, 1) returns [ 2 ].
• map:find(map { 1: map { 2: map { 3: 4 } } }, 3) returns [ 4 ].
• map:find((1, 'b', true#0), 1) returns an empty array.

map:for-each

Signatures
map:for-each($map as map(*), $function as function(xs:anyAtomicType, item()*) as item()*) as item()*

Summary
Applies the specified $function to every key/value pair of the supplied $map and returns the results as a sequence.

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($key, $value) { $key + $value }
)

map:get

Signatures
map:get($map 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 that has a key equal to the supplied value of $key. If there is such an entry, the function 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. A return value of () from map:get could indicate that the key is present in the map with an associated value of (), or it could indicate that the key is not present in the map. The two cases can be distinguished by calling map:contains. Invoking the map as a function item has the same effect as calling get: that is, when $map is a map, the expression $map($K) is equivalent to get($map, $K). Similarly, the expression get(get(get($map, 'employee'), 'name'), 'first') can be written as get(get(get($map, 'employee'), 'name'), 'first').

Examples
• map:get($week, 4) returns "Thu".
• map:get($week, 9) returns (). (When the key is not present, the function returns an empty sequence.).
• map:get(map:entry(7, ()), 7) returns (). (An empty sequence as the result can also signify that the key is present and the associated value is an empty sequence.).
### map:keys

**Signatures**

```
map:keys($map as map(*)) as xs:anyAtomicType*
```

**Summary**

Returns a sequence containing all the key values present in a map. The function takes the supplied `$map` 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($maps as map(*)*) as map(*),
map:merge($maps as map(*)*, $options as map(*)) as map(*),
```

**Summary**

Constructs and returns a new map. The map is formed by combining the contents of the supplied `$maps`. The maps are combined as follows:

1. There is one entry in the new map for each distinct key present in the union of the input maps.
2. The `$options` argument defines how duplicate keys are handled. Currently, a single option `duplicates` exists, and its allowed values are `use-first`, `use-last`, `combine` and `reject` (default: `use-first`).

**Examples**

- `map:merge(())` creates an empty map.
- `map:merge((map:entry(0, "no"), map:entry(1, "yes")))` creates `map { 0: "no", 1: "yes" }`.
- The following function adds a seventh entry to an existing map:
  ```
  map:merge(($week, map { 7: "---" }))
  ```
- In the following example, the values of all maps are combined, resulting in a map with a single key `map { "key": (1, 2, 3) }`:
  ```
  map:merge(
    for $i in 1 to 3 return map { 'key': $i },
    map { 'duplicates': 'combine' }
  )
  ```

### map:put

**Signatures**

```
map:put($map as map(*), $key as xs:anyAtomicType, $value as item()* as map(*)
```

**Summary**

Creates a new map, containing the entries of the supplied `$map` and a new entry composed by `$key` and `$value`. The semantics of this function are equivalent to `map:merge((map { $key, $value }, $map))`.

### map:remove

**Signatures**

```
map:remove($map as map(*), $keys as xs:anyAtomicType*) as map(*).
```

**Summary**

Constructs a new map by removing entries from an existing map. The entries in the new map correspond to the entries of `$map`, excluding entries supplied via `$keys`. No failure occurs if the input map contains no entry with the supplied keys; the input map is returned unchanged.

**Examples**

**map:size**

**Signatures**

map:size($map as map(*)) as xs:integer.

**Summary**

Returns a the number of entries in the supplied map. The function takes the supplied $map 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.

**Changelog**

Version 8.6

- Added: map:find

- Updated: map:merge: Signature extended with options argument. By default, value of first key is now adopted (instead of last, as in previous versions).

Version 8.4

- Removed: map:serialize (use fn:serialize instead)

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 56. Math Module

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 $\pi$ whose lexical representation is 3.141592653589793.

**Examples**

- $2\times$math:pi() returns 6.283185307179586e0.
- $60 \times (\text{math:pi() div 180})$ converts an angle of 60 degrees to radians.

math:sqrt

**Signatures**

math:sqrt($\text{arg as xs:double}$?) as xs:double?

**Summary**

Returns the square root of $\text{arg}$. If $\text{arg}$ is the empty sequence, the empty sequence is returned. Otherwise the result is the xs:double value of the mathematical square root of $\text{arg}$.

math:sin

**Signatures**

math:sin($\text{arg as xs:double}$?) as xs:double?

**Summary**

Returns the sine of the $\text{arg}$, expressed in radians. If $\text{arg}$ is the empty sequence, the empty sequence is returned. Otherwise the result is the sine of $\text{arg}$, treated as an angle in radians.

math:cos

**Signatures**

math:cos($\text{arg as xs:double}$?) as xs:double?

**Summary**

Returns the cosine of $\text{arg}$, expressed in radians. If $\text{arg}$ is the empty sequence, the empty sequence is returned. Otherwise the result is the cosine of $\text{arg}$, treated as an angle in radians.

math:tan

**Signatures**

math:tan($\text{arg as xs:double}$?) as xs:double?

**Summary**

Returns the tangent of $\text{arg}$, expressed in radians. If $\text{arg}$ is the empty sequence, the empty sequence is returned. Otherwise the result is the tangent of $\text{arg}$, treated as an angle in radians.

math:asin

**Signatures**

math:asin($\text{arg as xs:double}$?) 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**

```xml
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**

```xml
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**

```xml
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**

```xml
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**

```xml
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**

```xml
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**

```xml
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:log10(100)` returns 2.
Additional Functions

**math:e**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:e() as xs:double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the xs:double value of the mathematical constant e whose lexical representation is 2.718281828459045.</td>
</tr>
<tr>
<td>Examples</td>
<td>• 5*math:e() returns 13.591409142295225.</td>
</tr>
</tbody>
</table>

**math:sinh**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:sinh($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>• math:sinh(0) returns 0.</td>
</tr>
</tbody>
</table>

**math:cosh**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:cosh($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>• math:cosh(0) returns 1.</td>
</tr>
</tbody>
</table>

**math:tanh**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:tanh($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>• math:tanh(100) returns 1.</td>
</tr>
</tbody>
</table>

**math:crc32**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:crc32($string as xs:string?) as xs:hexBinary?.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Calculates the CRC32 check sum of the given $string. If an empty sequence is supplied, the empty sequence is returned.</td>
</tr>
<tr>
<td>Examples</td>
<td>• math:crc32(&quot;&quot;) returns '00000000'.</td>
</tr>
</tbody>
</table>

  • math:crc32("BaseX") returns '4C06FC7F'. |

**Changelog**

Version 9.1

- Updated: math:crc32 can be called with empty sequence.

Version 7.5

- Moved: math:random and math:uuid have been moved to the Random Module.

Version 7.3

- Added: math:crc32 and math:uuid have been adopted from the obsolete Utility Module.
Chapter 57. Process Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for executing system commands from XQuery.

Conventions

All functions and errors in this module are assigned to the http://basex.org/modules/proc namespace, which is statically bound to the proc 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*, $options as map(xs:string, 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 $options parameter contains process options:

- encoding: convert result to the specified encoding. If no encoding is supplied, the system’s default encoding is used.
- timeout: abort process execution after the specified number of seconds.
- dir: process command in the specified directory.
- input: standard string input (stdin) to be passed on to the command.

Errors

encoding: the specified encoding does not exist or is not supported.
timeout: the specified timeout was exceeded.
error: the command could not be executed, or an I/O exception was raised.

code....: If the commands returns an exit code different to 0, an error will be raised. Its code will consist of the letters code and four digits with the exit code.

Examples

- proc:system('date') returns the current date on a Linux system.
- Analyses the given input and counts the number of lines, words and characters (provided that wc is available on the system):

  proc:system(
    'wc', (),
    map ( 'input': 'A B' || out:nl() || 'C' )
  )
- The following example returns “Command not found” (unless xyz is a valid command on the system):

  try {
    proc:system('xyz')
  } catch proc:error {
    'Command not found: ' || $err:description
  }

proc:execute

Signatures

proc:execute($cmd as xs:string) as element(result),
proc:execute($cmd as xs:string, $args as xs:string*) as
### Process Module

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
</table>
| `element(result), proc:execute($cmd as xs:string, $args as xs:string*, $options as map(xs:string, xs:string)) as element(result)` | Executes the specified command in a separate process and returns the result as element:  
  - `$cmd` is the name of the command, and arguments to the command may be specified via `$args`.  
  - The same `$options` are allowed as for `proc:system`.  
  - Instead of the `proc:$error` error, the error message and process code will be assigned to the returned elements.  
  - Instead of the `proc:code...` error, the error message will be assigned to the returned element (no process code will be returned).  
  The result has the following structure:  
  ```xml  
  <result>  
  <output>...output...</output>  
  <error>...error message...</error>  
  <code>...process code...</code>  
  </result>  
  ```  
| Errors | encoding: the specified encoding does not exist or is not supported.  
|timeout: the specified timeout was exceeded. |
| 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. |

### proc:fork

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proc:fork($cmd as xs:string) as element(result), proc:fork($cmd as xs:string, $args as xs:string*) as element(result), proc:fork($cmd as xs:string, $args as xs:string*, $options as map(xs:string, xs:string)) as element(result)</code></td>
<td>Executes the specified command and ignores the result. <code>$cmd</code> is the name of the command, and arguments to the command may be specified via <code>$args</code>. The same <code>$options</code> are allowed as for <code>proc:system</code> (but the encoding will be ignored).</td>
</tr>
<tr>
<td>Errors</td>
<td>encoding: the specified encoding does not exist or is not supported.</td>
</tr>
<tr>
<td>Examples</td>
<td></td>
</tr>
</tbody>
</table>
- `proc:fork('sleep', '5')`: sleep for 5 seconds (no one should notice). |

### proc:property

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proc:property($name as xs:string) as xs:string?</code></td>
<td>Returns the system property, specified by <code>$name</code>, or a context parameter of the <code>web.xml</code> file with that name (see Web Applications). An empty sequence is returned if the property does not exist. For environment variables of the operating system, please use <code>fn:environment-variable</code>.</td>
</tr>
<tr>
<td>Examples</td>
<td></td>
</tr>
</tbody>
</table>
- `proc:property('java.class.path')` returns the full user class path.  
- `map:merge(proc:property-names() ! map:entry(.,'map:entry(., proc:property(.))'))` returns a map with all system properties. |

### proc:property-names

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proc:property-names() as xs:string*</code></td>
<td></td>
</tr>
</tbody>
</table>
Summary
Returns the names of all Java system properties and context parameters of the web.xml file (see Web Applications). For environment variables of the operating system, please use fn:available-environment-variables.

Examples
- proc:property('java.runtime.version') returns the version of the Java runtime engine.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code...</td>
<td>The result of a command call with an exit code different to 0.</td>
</tr>
<tr>
<td>code9999</td>
<td>A command could not be executed.</td>
</tr>
<tr>
<td>encoding</td>
<td>The specified encoding does not exist or is not supported.</td>
</tr>
<tr>
<td>timeout</td>
<td>The specified timeout was exceeded.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.0
- Added: proc:fork
- Updated: error codes; errors now use the module namespace
- Updated: new input option; revised error handling

Version 8.6
- Updated: proc:system, proc:exec: encoding option moved to options argument, timeout and dir options added.

Version 8.3
- Added: proc:property, proc:property-names.

The module was introduced with Version 7.3.
Chapter 58. Profiling Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains various functions to test and profile code, and to dump information to standard output.

Conventions

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

Performance Functions

prof:track

Signatures

prof:track($expression as item()) as item()*, prof:track($expression as item(), $options as map(*)?) as item()*

Summary

Measures the execution time and memory consumption required for evaluating the specified $expression and returns a map with the results. The following $options are available:

• time: Include execution time in result as xs:decimal (unit: milliseconds; default: true).

• memory: Include memory consumption in result as xs:integer (unit: bytes; default: false).

• value: Include value in result (default: true).

Helpful notes:

• If you are not interested in some of the returned results, you should disable them to save time and memory.

• Profiling might change the execution behavior of your code: An expression that might be executed iteratively will be cached by the profiling function.

• If a value has a compact internal representation, memory consumption will be very low, even if the serialized result may consume much more memory.

• Please note that memory profiling is only approximative, so it can be quite misleading. If the memory option is enabled, main-memory will be garbage-collected before and after evaluation to improve the quality of the measurement.

Properties

The function is non-deterministic: evaluation order will be preserved by the compiler.

Examples

• Return a human-readable representation of the memory consumption caused by fetching an XML document. fetch:doc is used, as fn:doc may already be evaluated at compilation time:

prof:track(fetch:doc('factbook.xml'))?memory

=> prof:human()

• The function call prof:track((1 to 1000000)[. mod 2 = 0], map { 'time': false() }) will return something similar to:

map {
    "memory": 21548400,
    "value": (2, 4, 6, 8, 10, ...)
}
Profiling Module

prof:time
Signatures prof:time($expr as item()) as item()*, prof:time($expr as item(), $label as xs:string) as item()*
Summary Measures the time needed to evaluate $expr and outputs a string to standard error or, if the GUI is used, to the Info View. An optional $label may be specified to tag the profiling result. See prof:track for further notes.
Properties The function is non-deterministic: evaluation order will be preserved by the compiler.
Examples • prof:time(prof:sleep(1000)) outputs something similar to 1000.99 ms.

prof:memory
Signatures prof:memory($expr as item()) as item()*, prof:memory($expr as item(), $label as xs:string) as item()*
Summary Measures the memory allocated by evaluating $expr and outputs a string to standard error or, if the GUI is used, to the Info View. An optional $label may be specified to tag the profiling result. See prof:track for further notes.
Properties The function is non-deterministic: evaluation order will be preserved by the compiler.
Examples • prof:memory((1 to 100000) ! <a/>) will output something similar to 5620 kB.

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 and returns different values every time it is called. Its evaluation order will be preserved by the compiler.
Examples • convert:integer-to-dateTime(prof:current-ms()) returns the current milliseconds in the xs:dateTime format.

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 and returns different values every time it is called. Its evaluation order will be preserved by the compiler.
Examples Measures the time of an expression:
let $ns1 := prof:current-ns()
return ( (: process to measure :) (1 to 1000000)[. = 0],
let $ns2 := prof:current-ns()
let $ms := (((($ns2 - $ns1) idiv 10000) div 100)
return $ms || ' ms'
)

Debugging Functions
prof:dump
Signatures prof:dump($expr as item()) as empty-sequence(), prof:dump($expr as item()*, $label as xs:string) as empty-sequence().
### Profiling Module

<table>
<thead>
<tr>
<th>Summary</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>In contrast to fn:trace(), the consumed expression will not be passed on.</td>
</tr>
</tbody>
</table>

**prof:variables**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>prof:variables() as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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 INLINELIMIT to 0.</td>
</tr>
<tr>
<td>Properties</td>
<td>The function is non-deterministic: evaluation order will be preserved by the compiler.</td>
</tr>
<tr>
<td>Examples</td>
<td>• for $x$ in 1 to 2 return ($x$, prof:variables()) will dump the values of $x$ to standard error.</td>
</tr>
</tbody>
</table>

**prof:type**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>prof:type($expr$ as item()<em>) as item()</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

**prof:gc**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>prof:gc() as empty-sequence(), prof:gc($count$ as xs:integer) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Enforces Java garbage collection. If no $count$ is supplied, garbage will be collected once. Please note that this function should only be used for debugging purposes; in productive code, it is best to trust the garbage collecting strategies of Java.</td>
</tr>
</tbody>
</table>

**prof:runtime**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>prof:runtime($name$ of xs:string) as xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the value of the specified runtime $option$. The following options exist:</td>
</tr>
<tr>
<td></td>
<td>• max: Maximum memory that the Java virtual machine will attempt to use.</td>
</tr>
<tr>
<td></td>
<td>• total: Total memory in the Java virtual machine (varies over time).</td>
</tr>
<tr>
<td></td>
<td>• used: Currently used memory (varies over time, will shrink after garbage collection).</td>
</tr>
<tr>
<td></td>
<td>• processors: number of processors available to the Java virtual machine.</td>
</tr>
<tr>
<td>option</td>
<td>The specified option is unknown.</td>
</tr>
<tr>
<td>Examples</td>
<td>• prof:gc(3), prof:human(prof:runtime('used')) performs some garbage collection and returns the currently used amount of memory in a user-friendly format.</td>
</tr>
</tbody>
</table>

**Helper Functions**

**prof:void**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>prof:void($value$ as item()*) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Properties</td>
<td>The function is non-deterministic: evaluation order will be preserved by the compiler.</td>
</tr>
</tbody>
</table>
Examples

• prof:void(fetch:binary('http://my.rest.service')) performs an HTTP request and ignores the result.

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.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
<td>The specified option is unknown.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.2

• Added: prof:gc, prof:runtime
• Updated: prof:track: decimal timing results; by default no memory profiling

Version 9.0

• Added: prof:track
• Updated: renamed prof:mem to prof:memory, prof:time: $cache argument removed

Version 8.5

• Added: prof:type (moved from XQuery Module)

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 59. Random Module

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 and errors in this module are assigned to the http://basex.org/modules/random namespace, which is statically bound to the random 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
bounds: 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
bounds: the maximum value is out of bounds. negative: 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:seeded-permutation

Signatures  random:seeded-permutation($seed as xs:integer, $items as item()*) as item()*

Summary  Returns a random permutation of the specified $items. The random order is created using the initial seed given in $seed.

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounds</td>
<td>The specified maximum value is out of bounds.</td>
</tr>
<tr>
<td>negative</td>
<td>The specified number of values to be returned is negative.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.0

• Updated: error codes updated; errors now use the module namespace

Version 8.5

• Added: random:seeded-permutation

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 60. Repository Module

This XQuery Module contains functions for installing, listing and deleting modules contained in the Repository.

Conventions

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

Functions

repo:install

Signatures: repo:install($uri as xs:string) as empty-sequence().

Summary: Retrieves and installs a package from the given $uri. Existing packages are replaced.

Errors:
- not-found: a package does not exist.
- descriptor: the package descriptor is invalid.
- installed: the module contained in the package to be installed is already installed as part of another package.
- parse: an error occurred while parsing the package.
- version: the package version is not supported.

repo:delete

Signatures: repo:delete($package as xs:string) as empty-sequence().

Summary: Deletes a $package. The argument contains the package name, optionally suffixed with a dash and the package version.

Errors:
- not-found: a package does not exist.
- delete: the package cannot be deleted.

repo:list

Signatures: repo:list() as element(package)*.

Summary: Lists the names and versions of all currently installed packages.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>The package cannot be deleted because of dependencies, or because files are missing.</td>
</tr>
<tr>
<td>descriptor</td>
<td>The package descriptor is invalid.</td>
</tr>
<tr>
<td>installed</td>
<td>The module contained in the package to be installed is already installed as part of another package.</td>
</tr>
<tr>
<td>not-found</td>
<td>A package does not exist.</td>
</tr>
<tr>
<td>parse</td>
<td>An error occurred while parsing the package.</td>
</tr>
<tr>
<td>version</td>
<td>The package version is not supported.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.0

- Updated: error codes updated; errors now use the module namespace
Repository Module

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 61. 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 mostly useful when building Web Applications.

The module is based on the EXQuery Request Module draft.

Conventions

• The module will be available if the basex-api library is found in the classpath. This is the case if you use one of the complete distributions of BaseX (zip, exe, war).

• All functions are assigned to the http://exquery.org/ns/request namespace, which is statically bound to the request prefix.

• If any of the functions is called outside the servlet context, basex:http is raised.

The following example illustrated what components a URI may consist of (the example is derived from RFC 3986):

```
foo://example.com:8042/over/there?name=ferret&code=7
```

<table>
<thead>
<tr>
<th>scheme</th>
<th>hostname</th>
<th>port</th>
<th>path</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>over/there</td>
<td>?name=ferret&amp;code=7</td>
</tr>
</tbody>
</table>

General Functions

request:method

Signatures  request:method() as xs:string
Summary     Returns the Method of the HTTP request.

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.
### Request Module

<table>
<thead>
<tr>
<th>Example</th>
<th>For the example given in the introduction, this function would return 8042.</th>
</tr>
</thead>
</table>

#### request:path

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:path() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the Path component of the URI of an HTTP request.</td>
</tr>
<tr>
<td>Example</td>
<td>For the example given in the introduction, this function would return /over/there.</td>
</tr>
</tbody>
</table>

#### request:query

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:query() as xs:string?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the Query component of the URI of an HTTP request. If no query component exists, an empty sequence is returned.</td>
</tr>
<tr>
<td>Example</td>
<td>For the example given in the introduction, this function would return name=ferret&amp;code=7.</td>
</tr>
</tbody>
</table>

#### request:uri

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:uri() as xs:anyURI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the complete URI of an HTTP request as it has been specified by the client.</td>
</tr>
<tr>
<td>Example</td>
<td>For the example given in the introduction, this method would return foo://example.com:8042/over/there?name=ferret&amp;code=7.</td>
</tr>
</tbody>
</table>

#### request:context-path

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:context-path() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the context of the request. For servlets in the default (root) context, this method returns an empty string.</td>
</tr>
</tbody>
</table>

### Connection Functions

#### request:address

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:address() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the IP address of the server.</td>
</tr>
</tbody>
</table>

#### request:remote-hostname

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:remote-hostname() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the fully qualified hostname of the client that sent the request.</td>
</tr>
</tbody>
</table>

#### request:remote-address

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:remote-address() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the IP address of the client that sent the request.</td>
</tr>
</tbody>
</table>

#### 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.

Errors
parameter: the request has invalid parameters.

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('code') would return 7.

Errors
parameter: the request has invalid parameters.

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**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:cookie($name as xs:string) as xs:string?, request:cookie($name as xs:string, $default as xs:string) as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Attribute Functions

**request:attribute-names**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:attribute-names() as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the names of all HTTP request attributes.</td>
</tr>
</tbody>
</table>

**request:attribute**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:attribute($name as xs:string) as item()<em>, request:attribute($name as xs:string, $default as item()</em>) as item()*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the value of an attribute of the HTTP request. If the attribute does not exist, an empty sequence or the optionally specified default value is returned instead.</td>
</tr>
</tbody>
</table>
| 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. |

**request:set-attribute**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>request:set-attribute($name as xs:string, $value as item()*) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Binds the specified $value to the request attribute with the specified $name.</td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter</td>
<td>Request has invalid parameters.</td>
</tr>
</tbody>
</table>

### Changelog

**Version 9.3**

- Added: `request:attribute-names`, `request:set-attribute`

**Version 7.9**

- Updated: `request:attribute`: return type generalized, default argument added

**Version 7.8**

- Added: `request:context-path`
Version 7.7

• Added: request:attribute

This module was introduced with Version 7.5.
Chapter 62. RESTXQ Module

Conventions

- The module will be available if the basex-api library is found in the classpath. This is the case if you use one of the complete distributions of BaseX (zip, exe, war).

- All functions are assigned to the http://exquery.org/ns/restxq namespace, which is statically bound to the rest prefix.

- 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, basex:http is raised.

General Functions

rest:base-uri

Signatures | rest:base-uri() as xs:anyURI
Summary     | Returns the implementation-defined base URI of the resource function.

rest:uri

Signatures | rest:uri() as xs:anyURI
Summary     | 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     | Returns a WADL description of all available REST services.

rest:init

Signatures | rest:init() as empty-sequence(), rest:init($update as xs:boolean) as empty-sequence()
Summary     | Initializes the RESTXQ module cache:
            | - By default, the cache will be discarded, and all modules will be parsed and cached again.
            | - If $update is enabled, the background caching behavior is simulated (see PARSERESTXQ): Only updated modules will be parsed.
            | - This function should be called if new RESTXQ code is deployed at runtime.

Changelog

Version 9.4

- Updated: rest:init argument added
Version 8.6

• Added: rest:init

This module was introduced with Version 7.7.
Chapter 63. 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 module will be available if the basex-api library is found in the classpath. This is the case if you use one of the complete distributions of BaseX (zip, exe, war).

- All functions and errors are assigned to the http://basex.org/modules/session namespace, which is statically bound to the session prefix.

- If any of the functions is called outside the servlet context, basex:http is raised.

- As sessions are side-effecting operations, all functions are flagged as non-deterministic. As a result, some query optimizations will be suppressed.

Functions

**session:id**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>session:id() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the session ID of a servlet request.</td>
</tr>
<tr>
<td>Errors</td>
<td>not-found: No session is available for the current client.</td>
</tr>
<tr>
<td>Examples</td>
<td>Running the server-side XQuery file id.xq via <a href="http://localhost:8984/id.xq">http://localhost:8984/id.xq</a>:</td>
</tr>
</tbody>
</table>

```
import module namespace session = "http://basex.org/modules/session";
'Session ID: ' || session:id()
```

**session:created**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>session:created() as xs:dateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the creation time of a session.</td>
</tr>
<tr>
<td>Errors</td>
<td>not-found: No session is available for the current client.</td>
</tr>
</tbody>
</table>

**session:accessed**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>session:accessed() as xs:dateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the last access time of a session.</td>
</tr>
<tr>
<td>Errors</td>
<td>not-found: No session is available for the current client.</td>
</tr>
</tbody>
</table>

**session:names**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>session:names() as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the names of all attributes bound to the current session.</td>
</tr>
<tr>
<td>Examples</td>
<td>Running the server-side XQuery file names.xq via <a href="http://localhost:8984/names.xq">http://localhost:8984/names.xq</a>:</td>
</tr>
</tbody>
</table>

```
import module namespace session = "http://basex.org/modules/session";
session:names() ! element variable { . }
```
Session Module

**session:get**

**Signatures**

```
session:get($name as xs:string) as item()*, session:get($name as xs:string, $default as item())* as item()*
```

**Summary**

Returns the value of a session attribute with the specified $name. If the attribute is unknown, an empty sequence or the optionally specified $default value will be returned instead.

**Examples**

Running the server-side XQuery file `get.xq` via `http://localhost:8984/get.xq?key=user`:

```xquery
import module namespace session = "http://basex.org/modules/session";
'Value of ' || $key || ': ' || session:get($key)
```

**session:set**

**Signatures**

```
session:set($name as xs:string, $value as item()) as empty-sequence()
```

**Summary**

Binds the specified $value to the session attribute with the specified $name.

**Errors**

`not-found`: No session is available for the current client.

**Examples**

Running the server-side XQuery file `set.xq` via `http://localhost:8984/set.xq?key=user&value=john`:

```xquery
import module namespace session = "http://basex.org/modules/session";
session:set($key, $value), 'Variable was set.'
```

**session:delete**

**Signatures**

```
session:delete($name as xs:string) as empty-sequence()
```

**Summary**

Deletes a session attribute with the specified $name.

**Examples**

Running the server-side XQuery file `delete.xq` via `http://localhost:8984/delete.xq?key=user`:

```xquery
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**
---|---
`not-found` | No session is available for the current client.

**Changelog**

**Version 9.4**

- Updated: Only create session if required.

**Version 9.3**

- Updated: `session:get`: Values that have no XQuery type will be returned as strings.
Version 9.0

- Updated: error codes updated; errors now use the module namespace

Version 8.0

- Updated: Allow sequences as session values.

This module was introduced with Version 7.5.
Chapter 64. 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 module will be available if the basex-api library is found in the classpath. This is the case if you use one of the complete distributions of BaseX (zip, exe, war).
- All functions and errors are assigned of the http://basex.org/modules/sessions namespace, which is statically bound to the sessions prefix.
- If any of the functions is called outside the servlet context, basex:http is raised.
- If a specified session id is not found, not-found is raised.
- As sessions are side-effecting operations, all functions are flagged as non-deterministic. As a result, some query optimizations will be suppressed.

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 attributes bound to the session specified by $id.

sessions:get

Signatures  | sessions:get($id as xs:string, $name as xs:string) as item()*, sessions:get($id as xs:string, $name as xs:string, $default as item()*) as item()*
Summary     | Returns the value of an attribute with the specified $name from the session with the specified $id. If the attribute is unknown, an empty sequence or the optionally specified $default value will be returned instead.
sessions:set

Signatures
sessions:set($id as xs:string, $name as xs:string, $value as item()* as empty-sequence())

Summary
Returns the specified value to the attribute with the specified $name from the session with the specified $id.

sessions:delete

Signatures
sessions:delete($id as xs:string, $name as xs:string) as empty-sequence()

Summary
Deletes an attribute with the specified $name from the session with the specified $id.

sessions:close

Signatures
sessions:close($id as xs:string) as empty-sequence()

Summary
Unregisters the session specified by $id.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>not-found</td>
<td>The specified session is not available.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.3
• Updated: sessions:get: Values that have no XQuery type will be returned as strings.

Version 9.0
• Updated: error codes updated; errors now use the module namespace

Version 8.4
• Updated: Allow sequences as session values.

This module was introduced with Version 7.5.
Chapter 65. SQL Module

This XQuery Module contains functions to access relational databases from XQuery using SQL. With this module, you can execute query, update, and prepared statements, and the result sets are returned as sequences of XML elements representing tuples. Each element has children representing the columns returned by the SQL statement.

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 and errors in this module are assigned to the http://basex.org/modules/sql namespace, which is statically bound to the sql 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**

init: the specified driver is not found.

**sql:connect**

**Signatures**

```
sql:connect($url as xs:string) as xs:anyURI,
sql:connect($url as xs:string, $user as xs:string, $password as xs:string) as xs:anyURI,
sql:connect($url as xs:string, $user as xs:string, $password as xs:string, $options as map(*)?) as xs:anyURI,
```

**Summary**

This function establishes a connection to a relational database and returns a connection id. 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.

**Errors**

error: an SQL exception occurred when connecting to the database.

**Examples**

Connects to an SQL Server and sets autocommit to true:

```
sql:connect('jdbc:sqlserver://DBServer', map { 'autocommit': true() })
```

**sql:execute**

**Signatures**

```
sql:execute($id as xs:anyURI, $statement as xs:string) as item()*,
sql:execute($id as xs:anyURI, $statement as xs:string, $options as map(*)?) as item()*
```

**Summary**

This function executes an SQL $statement, using the connection with the specified $id. The returned result depends on the kind of statement:

- If an update statement was executed, the number of updated rows will be returned as integer.
- Otherwise, an XML representation of all results will be returned.

With $options, the following parameter can be set:
timeout: query execution will be interrupted after the specified number of seconds.

**Errors**

- error: an error occurred while executing SQL.
- id: the specified connection does not exist.
- timeout: query execution exceeded timeout.

### sql:execute-prepared

**Signatures**

```xml
sql:execute-prepared($id as xs:anyURI, $params as element(sql:parameters)) as item()*,
sql:execute-prepared($id as xs:anyURI, $params as element(sql:parameters), $options as map(*)?) as item()*
```

**Summary**

This function executes a prepared statement with the specified $id. The output format is identical to sql:execute. 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:

```xml
element sql:parameters {
    element sql:parameter {
        attribute type { "bigdecimal" | "boolean" | "byte" | "date" |
        "double" | "float" |
        "int" | "long" | "short" | "sqlxml" | "string" | "time" |
        "timestamp" },
        attribute null { "true" | "false" }?,
        text *
    }
}
```

With $options, the following parameter can be set:

- timeout: query execution will be interrupted after the specified number of seconds.

**Errors**

- attribute: an attribute different from type and null is set for a <sql:parameter/> element.
- error: an error occurred while executing SQL.
- id: the specified connection does not exist.
- parameters: no parameter type specified.
- timeout: query execution exceeded timeout.
- type: the value of a parameter cannot be converted to the specified format.

### sql:prepare

**Signatures**

```xml
sql:prepare($id as xs:anyURI, $statement as xs:string) as xs:anyURI
```

**Summary**

This function prepares an SQL $statement, using the specified connection $id, and returns the id reference to this statement. The statement is a string with one or more '?' placeholders. If the value of a field has to be set to NULL, then the attribute null of the <sql:parameter/> element must be true.

**Errors**

- error: an error occurred while executing SQL.
- id: the specified connection does not exist.

### sql:commit

**Signatures**

```xml
sql:commit($id as xs:anyURI) as empty-sequence()
```

**Summary**

This function commits the changes made to a relational database, using the specified connection $id.

**Errors**

- error: an error occurred while executing SQL.
- id: the specified connection does not exist.

### sql:rollback

**Signatures**

```xml
sql:rollback($id as xs:anyURI) as empty-sequence()
```

**Summary**

This function rolls back the changes made to a relational database, using the specified connection $id.
SQL Module

Errors
error: an error occurred while executing SQL
id: the specified connection does not exist.

sql:close

Signatures
sql:close($id as xs:anyURI) as empty-sequence()

Summary
This function closes a database connection with the specified $id. Opened connections will automatically be closed after the XQuery expression has been evaluated, but in order to save memory, it is always recommendable to close connections that are not used anymore.

Errors
error: an error occurred while executing SQL
id: the specified connection does not exist.

Examples

Direct queries
A simple select statement can be executed as follows:

let $id := sql:connect("jdbc:postgresql://localhost:5432/coffeehouse")
return sql:execute($id, "SELECT * FROM coffees WHERE price < 10")

The result may 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">3.5</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>

Prepared Statements
A prepared select statement can be executed in the following way:

let $conn := sql:connect("jdbc:postgresql://localhost:5432/coffeehouse")
let $prep := sql:prepare($conn, "SELECT * FROM coffees WHERE price < ? AND cof_name = ?")
let $params := <sql:parameters>
  <sql:parameter type='double'>10</sql:parameter>
  <sql:parameter type='string'>French_Roast</sql:parameter>
</sql:parameters>
return sql:execute-prepared($prep, $params)

SQLite
The following expression demonstrates how SQLite can be addressed with the Xerial SQLite JDBC driver:

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")  
)

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attribute</td>
<td>An attribute different from type and null is set for a <a href="">sql:parameter/</a> element.</td>
</tr>
<tr>
<td>error</td>
<td>An SQL exception occurred.</td>
</tr>
<tr>
<td>id</td>
<td>A connection does not exist.</td>
</tr>
<tr>
<td>init</td>
<td>A database driver is not found.</td>
</tr>
<tr>
<td>parameters</td>
<td>No parameter type specified.</td>
</tr>
<tr>
<td>timeout</td>
<td>Query execution exceeded timeout.</td>
</tr>
<tr>
<td>type</td>
<td>The value of a parameter cannot be converted to the specified format.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.6

- Updated: sql:execute-prepared, additional types added

Version 9.0

- Updated: sql:execute, sql:execute-prepared: Return update count for updating statements. $options argument added.
- Updated: Connection ids are URIs now.
- Updated: error codes updated; errors now use the module namespace

Version 7.5

- Updated: prepared statements are now executed via sql:execute-prepared

The module was introduced with Version 7.0.
Chapter 66. Store Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions to organize values in a persistent main-memory key-value store.

The store is useful if data (a system configuration, maps serving as indexes) needs to be repeatedly accessed. The store is persistent: Contents will be written to disk at shutdown time, and the serialized store will be retrieved from disk as soon as the store is used for the first time. The store will be stored in a binary store.basex file in the database directory.

In addition, custom stores can be read and written. Custom stores have filenames with the pattern store-NAME.basex. The implicit write of the standard store at shutdown time will be disabled if a custom store is used.

Functions of this module are non-deterministic and side-effecting: Updates will immediately be visible, and a repeated call of the same function may yield different results if the contents of the store have changed.

Conventions

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

Key-value operations

store:get

Signatures  
store:get($key as xs:string) as item()*.  

Summary  
Retrieves an entry from the store with the given $key. If the addressed entry does not exist, an empty sequence is returned.

store:put

Signatures  
store:put($key as xs:string, $value as item()*) as empty-sequence().  

Summary  
Stores an entry with the given $key and $value in the store:

- If the value is an empty sequence, the entry is removed.
- If a value refers to an opened database or is a lazy item, its contents are materialized in main memory.
- Values with function items are rejected.

store:get-or-put

Signatures  
store:get-or-put($key as xs:string, $put as function() as item()*) as item()*.  

Summary  
Retrieves an entry from the store with the given $key. The $put function will only be invoked if the entry does not exist, and its result will be stored and returned instead.

store:remove

Signatures  
store:remove($key as xs:string) as empty-sequence().  

Summary  
Removes an entry with the given $key from the store. No error will be raised if an addressed entry does not exist.
store:keys

Signatures: `store:keys() as xs:string*`

Summary: Lists the names of all keys.

store:clear

Signatures: `store:clear() as empty-sequence()`

Summary: Resets the store by removing all its entries.

Store Operations

store:read

Signatures: `store:read() as empty-sequence()`,

Summary: Retrieves the standard store from disk, or a custom store if a `$name` is supplied.

Errors:
- `io`: The store could not be read.
- `name`: The specified name is invalid.
- `not-found`: A store with the specified name does not exist.

store:write

Signatures: `store:write() as empty-sequence()`,

Summary: Writes the standard store to disk, or to a custom store file if a `$name` is supplied. If the standard store is empty, the store file will be deleted.

Errors:
- `io`: The store could not be written.
- `name`: The specified name is invalid.

store:list

Signatures: `store:list() as xs:string*`

Summary: Lists the names of all custom stores.

store:delete

Signatures: `store:delete($name as xs:string) as empty-sequence()`

Summary: Deletes a custom store from disk.

Errors:
- `name`: The specified name is invalid.
- `not-found`: A store with the specified name does not exist.

Examples

Use Case 1: Create/update a system configuration in a running BaseX server instance:

```xml
(: store an integer :)
store:put('version', 1),
(: retrieve existing or new value, store an element :)
let $license := store:get-or-put('license', function() { 'free' })
return store:put('info', <info>{$license = 'free' ?? 'Free' !! 'Professional' } License</info>),
(: store a map :)
store:put('data', map { 'year': 2022 }),
(: serialize configuration to disk :)
store:write()
```
The configuration can be requested by further operations, e.g. a client request:

```
store:get('version')
```

The store will still be available if BaseX is restarted until it is cleared.

**Use Case 2: Create index for fast lookup operations in the GUI:**

```
let $map := map:merge(
    for $country in db:get('factbook')/country
    for $religion in $country//religions
    group by $religion
    return map:entry($religion, data($country/@name))
)
return store:put('religions', $map)
```

A subsequent query can be used to access its contents:

```
store:get('religions')?Buddhism
```

Note that the store will eventually be written to disk unless it is invalidated before closing the GUI.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>io</td>
<td>The store could not be read or written.</td>
</tr>
<tr>
<td>name</td>
<td>The specified name is invalid.</td>
</tr>
<tr>
<td>not-found</td>
<td>A store with the specified name does not exist.</td>
</tr>
</tbody>
</table>

**Changelog**

The module was introduced with Version 10.
Chapter 67. String Module

Read this entry online in the BaseX Wiki.

Updated with Version 10: Renamed from Strings Module to String Module. The namespace URI has been updated as well.

This XQuery Module contains functions for string operations and computations.

Conventions

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

Computations

string:levenshtein

Signatures  
string:levenshtein($string1 as xs:string, $string2 as xs:string) as xs:double.

Summary  
Computes the Damerau-Levenshtein Distance for two strings and returns a double value (0.0 - 1.0). The returned value is computed as follows:

- 1.0 – distance / max(length of strings)
- 1.0 is returned if the strings are equal; 0.0 is returned if the strings are too different.

Examples

- string:levenshtein("flower", "flower") returns 1
- string:levenshtein("flower", "lewes") returns 0.5
- In the following query, the input is first normalized (words are stemmed, converted to lower case, and diacritics are removed). It returns 1:
  ```xquery
  let $norm := ft:normalize(?., map { 'stemming': true() })
  return string:levenshtein($norm("HOUSES"), $norm("house"))
  ```

string:soundex

Signatures  
string:soundex($string as xs:string) as xs:string.

Summary  
Computes the Soundex value for the specified string. The algorithm can be used to find and index English words with similar pronunciation.

Examples

- string:soundex("Michael") returns M240
- string:soundex("O'Brien") = string:soundex("O'Brien") returns true

string:cologne-phonetic

Signatures  
string:cologne-phonetic($string as xs:string) as xs:string.

Summary  
Computes the Kölner Phonetik value for the specified string. Similar to Soundex, the algorithm is used to find similarly pronounced words, but for the German language. As the first returned digit can be 0, the value is returned as string.

Examples

- string:cologne-phonetic("Michael") returns 645
- every $s in ("Mayr", "Maier", "Meier") satisfies string:cologne-phonetic($s) = "67" returns true
Formatting

The functions in this section have been adopted from the obsolete Output Module.

string:format

Signatures

string:format($format as xs:string, $items as item() ...) as xs:string

Summary

Returns a formatted string. The remaining arguments specified by $items are applied to the $format string, according to Java's printf syntax.

Errors

format: The specified format is not valid.

Examples

• string:format("%b", true()) returns true.
• string:format("%06d", 256) returns 000256.
• string:format("%e", 1234.5678) returns 1.234568e+03.

string:cr

Signatures

string:cr() as xs:string

Summary

Returns a single carriage return character (\r).

string:nl

Signatures

string:nl() as xs:string

Summary

Returns a single newline character (\n).

string:tab

Signatures

string:tab() as xs:string

Summary

Returns a single tabulator character (\t).

Changelog

Version 10.0

• Updated: Renamed from Strings Module to String Module. The namespace URI has been updated as well.

• Updated: string:format, string:cr, string:nl and string:tab adopted from the obsolete Output Module.

The Module was introduced with Version 8.3. Functions were adopted from the obsolete Utility and Output Modules.
Chapter 68. 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, automated way of testing software. The XUnit frameworks (such as SUnit or JUnit) allow testing of atomic units 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 be added'
  };

unit:before

Syntax

%unit:before, %unit:before(FUNCTION)

Summary

A function decorated with this annotation will be evaluated before each unit test as a separate transaction. 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 as a separate transaction. 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 as a separate transaction.

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 as a separate transaction.

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 fail: 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-equal function. Otherwise, raises an error. The default failure message can be overridden with the $info argument.
**Errors** | `fail`: 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** | `fail`: default error raised by this function.

**Example**

The following XQuUnit 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**

```xml
<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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail</td>
<td>An assertion failed, or an error was raised.</td>
</tr>
<tr>
<td>no-args</td>
<td>A test function must have no arguments.</td>
</tr>
<tr>
<td>private</td>
<td>A test function must not be private.</td>
</tr>
</tbody>
</table>
Changelog

Version 9.0
• Updated: error codes updated; errors now use the module namespace

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 69. Update Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides additional functions for performing updates and returning results in updating expressions.

Conventions

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

Except for update:output-cache, all functions are updating and thus comply to the XQuery Update constraints.

Updates

update:apply

**Signatures**

update:apply($function as function(*), $arguments as array(*)) as empty-sequence()

**Summary**
The updating variant of fn:apply applies the specified updating $function to the specified $arguments.

**Examples**

- Creates a new database with an initial document and adds a document to an existing database.

```xml
declare %updating function local:update(
    $database  as xs:string,
    $path      as xs:string,
    $function  as %updating function(item(), xs:string) as empty-sequence()
) as empty-sequence() {
    update:apply($function, [ $database, $path ])
};
local:update('new-db', 'doc.xml', db:create#2),
local:update('existing-db', 'doc.xml', db:add#2)
```

update:for-each

**Signatures**

update:for-each($seq as item()*, $function as function(item()) as item()*) as empty-sequence()

**Summary**
The updating variant of fn:for-each applies the specified updating $function to every item of $seq.

**Examples**

- Creates two databases:

```xml
let $names := ('db1', 'db2')
return update:for-each($names, db:create#1)
```

update:for-each-pair

**Signatures**

update:for-each-pair($seq1 as item()*, $function as function(item()) as item()*) as empty-sequence()

**Summary**
The updating variant of fn:for-each-pair applies the specified updating $function to the successive pairs of items of $seq1 and $seq2. Evaluation is stopped if one sequence yields no more items.

**Examples**

- Renames nodes in an XML snippets:
**Update Module**

```xml
copy $xml := <xml><a/>b/></xml>
modify update:for-each-pair('a', 'b'), ('d', 'e'),
  function($source, $target) {
    for $e in $xml/*[name() = $source]
      return rename node $e as $target
  }
return $xml
```

**update:map-for-each**

**Signatures**

update:map-for-each($map as map(*), $function as function(xs:anyAtomicType, item(*)*) as item(*)*)

**Summary**
The updating variant of map:for-each applies the specified $function to every key/value pair of the supplied $map and returns the results as a sequence.

**Examples**

- Inserts attributes into a document:

```xml
copy $doc := <xml/>
modify update:map-for-each(map {
    'id': 'id0',
    'value': 456
},
  function($key, $value) {
    insert node attribute { $key } { $value } into $doc
  }
}
return $doc
```

**Output**

**update:output**

**Signatures**

update:output($items as item(*)*) as empty-sequence()

**Summary**
This function can be used if MIXUPDATES is not enabled, and if values need to returned within an updating expression: The supplied $items will be cached and returned at the very end, i.e., after all updates on the pending update list have been processed. If one of the supplied items is affected by an update, a copy will be created and cached instead.

**Examples**

- `update:output("Prices have been deleted."), delete node //price` deletes all price elements in a database and returns an info message.

**update:cache**

**Signatures**

update:cache() as item(), update:cache($reset as xs:boolean) as item(*)

**Summary**
Returns the items that have been cached by update:output. The output cache can optionally be $reset. The function can be used to check which items will eventually be returned as result of an updating function. This function is non-deterministic: It will return different results before and after items have been cached. It is e.g. useful when writing unit tests.

**Changelog**

Version 9.3

- `update:cache:$reset` parameter added.
Version 9.1

- update:output: Maps and arrays can be cached if they contain no persistent database nodes or function items.

Version 9.0

- Updated: db:output renamed to update:output, db:output-cache renamed to update:cache

This module was introduced with Version 9.0.
Chapter 70. 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 and errors in this module are assigned to the http://basex.org/modules/user namespace, which is statically bound to the user prefix.

Read Operations

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 that are visible to the current user. |
| Examples | • After a fresh installation, user:list() will only return admin. |

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 that are visible to the current user. 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 | • After a fresh installation, user:list-details() returns output similar to the following one: |

```
<user name="admin" permission="admin">
  <password algorithm="digest">
    <hash>304bdfb0383c16f070a897fc1eb25cb4</hash>
  </password>
  <password algorithm="salted-sha256">
    <salt>871602799292195</salt>
    <hash>a065ca66fa3d6da5762c227587f1c8258c6dc08ee867e44a605a72da15dcb41</hash>
  </password>
</user>
```

| Errors | unknown: The specified username is unknown. |

user:exists

| Signatures | user:exists($name as xs:string) as xs:boolean. |
| Summary | Checks if a user with the specified $name exists. |

352
### Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>user:exists('admin')</code> will always yield true.</td>
<td></td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Errors</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: The specified username is invalid.</td>
<td></td>
</tr>
</tbody>
</table>

### user:check

#### Signatures

`user:check($name as xs:string, $password as xs:string) as empty-sequence()`,

#### Summary

Checks if the specified user and password is correct. Raises errors otherwise.

#### Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>user:check('admin',)</code> will raise an error if the password of the admin user is a non-empty string.</td>
<td></td>
</tr>
</tbody>
</table>

#### Errors

<table>
<thead>
<tr>
<th>Errors</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: The specified username is invalid.</td>
<td></td>
</tr>
<tr>
<td>unknown: The specified user does not exist.</td>
<td></td>
</tr>
<tr>
<td>password: The specified password is wrong.</td>
<td></td>
</tr>
</tbody>
</table>

### user:info

#### Signatures

`user:info() as element(info), user:info($name as xs:string) as element(info)`

#### Summary

Returns an info element, which may contain application-specific data. If a user $name is supplied, a user-specific element is returned. By default, the returned element has no contents. It can be modified via `user:update-info`.

#### Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>• After a fresh installation, <code>user:info()</code> returns <code>&lt;info/&gt;</code>.</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

### 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, $permissions as xs:string*) as empty-sequence(), user:create($name as xs:string, $password as xs:string, $permissions as xs:string*, $patterns as xs:string*) as empty-sequence(), user:create($name as xs:string, $password as xs:string, $permissions as xs:string*, $patterns as xs:string*, $info as element(info)) as empty-sequence()`

#### Summary

Creates a new user with the specified $name, $password, and $permissions:

- Local permissions are granted with non-empty glob $patterns.
- An $info element with application-specific information can be supplied.
- The default global permission (none) can be overwritten with an empty pattern or by omitting the last argument.
- Existing users will be overwritten.

#### Examples

<table>
<thead>
<tr>
<th>Examples</th>
<th>User Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <code>user:create('John', '7e$j#!1', 'admin')</code> creates a new user 'John' with admin permissions.</td>
<td></td>
</tr>
<tr>
<td>• <code>user:create('Jack', 'top!secret', 'read', 'index*')</code> creates a new user 'Jack' with read permissions for databases starting with the letters 'index'.</td>
<td></td>
</tr>
</tbody>
</table>
Errors  name: The specified username 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, $permissions as xs:string*) as empty-sequence(), user:grant($name as xs:string, $permissions as xs:string*, $patterns as xs:string*) as empty-sequence()`  

Summary  Grants global or local $permissions to a user with the specified $name. Local permissions are granted with non-empty glob $patterns.

Examples  
* `user:grant('John', 'create')` grants create permissions to the user 'John'.
* `user:grant('John', ('read','write'), ('index*','unit*'))` allows John to read all databases starting with the letters 'index', and to write to all databases starting with 'unit'.

Errors  unknown: The specified username is unknown.name: The specified username 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:drop**

Signatures  `user:drop($name as xs:string) as empty-sequence(), user:drop($name as xs:string, $patterns as xs:string*) as empty-sequence()`  

Summary  Drops a user with the specified $name. If non-empty glob $patterns are specified, only the database patterns will be removed.

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 username is unknown.name: The specified username 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.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:alter('John', 'Jack')` renames the user 'John' to 'Jack'.

Errors  unknown: The specified username is unknown.name: The specified username 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.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()`
User Module

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 username is unknown.name: The specified username is invalid.update: The operation can only be performed once per user or database pattern.

user:update-info

Signatures
user:update-info($info as element(info)) as empty-sequence(),
user:update-info($info as element(info), $name as xs:string) as empty-sequence()

Summary
Assigns the specified $info element to the user management or, if $name is supplied, to a specific user. This function can be used to manage application-specific data (groups, enhanced user info, etc.).

Examples
• Store initial groups information:

```
user:update-info(element info {
  for $group in ('editor', 'author', 'writer')
    return element group { $group }
})
```

• Add a group to a specific user:

```
user:update-info(<info group='editor'/>, 'john')
```

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>The &quot;admin&quot; user cannot be modified.</td>
</tr>
<tr>
<td>conflict</td>
<td>A user cannot be both altered and dropped.</td>
</tr>
<tr>
<td>equal</td>
<td>Name of old and new user is equal.</td>
</tr>
<tr>
<td>local</td>
<td>A local permission can only be 'none', 'read' or 'write'.</td>
</tr>
<tr>
<td>logged-in</td>
<td>The specified user is currently logged in.</td>
</tr>
<tr>
<td>name</td>
<td>The specified username is invalid.</td>
</tr>
<tr>
<td>password</td>
<td>The specified password is wrong.</td>
</tr>
<tr>
<td>pattern</td>
<td>The specified database name is invalid.</td>
</tr>
<tr>
<td>permission</td>
<td>The specified permission is invalid.</td>
</tr>
<tr>
<td>unknown</td>
<td>The specified user does not exist.</td>
</tr>
<tr>
<td>update</td>
<td>The operation can only be performed once per user or database pattern.</td>
</tr>
</tbody>
</table>

Changelog

Version 8.6
• Updated: user:create, user:info, user:update-info: $name parameter added.

Version 8.6
• Added: user:check, user:info, user:update-info.
• Updated: user:list, user:list-details: If called by non-admins, will only return the current user.

Version 8.4
• Updated: \texttt{user:create, user:grant, user:drop}: extended support for database patterns.

Version 8.1

• Added: \texttt{user:current}.

The Module was introduced with Version 8.0.
Chapter 71. Validation Module

This XQuery Module contains functions to perform validations against DTDs, XML Schema and RelaxNG. The documentation further describes how to use Schematron validation with BaseX.

Conventions

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

DTD Validation

Checks whether an XML document validates against a DTD. The input document can be specified as:

- `xs:string`, representing a URI (relative URIs will always be resolved against the static base URI of the query),
- `xs:string`, representing the resource in its string representation, or
- `node()`, representing the resource itself.

If no DTD is supplied in a function, the XML document is expected to contain an embedded DTD doctype declaration.

**validate:dtd**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>validate:dtd($input as item()) as empty-sequence(), validate:dtd($input as item(), $schema as xs:string?) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Validates the XML $input document against a $schema and returns an empty sequence or an error.</td>
</tr>
<tr>
<td>Errors</td>
<td>error: the validation fails. init: the validation process cannot be started. not-found: no DTD validator is available.</td>
</tr>
<tr>
<td>Examples</td>
<td>• validate:dtd('doc.xml', 'doc.dtd') validates the document doc.xml against the specified DTD file doc.dtd.</td>
</tr>
<tr>
<td></td>
<td>• The following example validates an invalid document against a DTD, which is specified as string:</td>
</tr>
<tr>
<td></td>
<td>try { let $doc := &lt;invalid/&gt; let $schema := ' &lt;!ELEMENT root (#PCDATA)&gt; ' return validate:dtd($doc, $schema)</td>
</tr>
<tr>
<td></td>
<td>} catch validate:error { 'DTD Validation failed.' }</td>
</tr>
</tbody>
</table>

**validate:dtd-info**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>validate:dtd-info($input as item()) as xs:string*, validate:dtd-info($input as item(), $schema as xs:string?) as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Validates the XML $input document against a $schema and returns warnings, errors and fatal errors in a string sequence.</td>
</tr>
<tr>
<td>Errors</td>
<td>init: the validation process cannot be started. not-found: no DTD validator is available.</td>
</tr>
</tbody>
</table>
**Examples**

- `validate:dtd-info(<invalid/>, '&lt;!ELEMENT root (#PCDATA)>&lt;/report>
  <status>invalid</status>
  <message level="Error" line="2" column="11">Element type "invalid" must be declared.</message>
</report>

**validate:dtd-report**

**Signatures**

`validate:dtd-report($input as item()) as element(report),
validate:dtd-report($input as item(), $schema as xs:string?) as element(report)`

**Summary**

Validates the XML `$input` document against a `$schema` and returns warnings, errors and fatal errors as XML.

**Errors**

- `init`: the validation process cannot be started.
- `not-found`: no DTD validator is available.

**Examples**

- `validate:dtd-report(<invalid/>, '&lt;!ELEMENT root (#PCDATA)>&lt;/report>
  <status>invalid</status>
  <message level="Error" line="2" column="11">Element type "invalid" must be declared.</message>
</report>`

**XML Schema Validation**

Checks whether an XML document validates against an XML Schema. The input document and the 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.

If no schema is given, the input is expected to contain an `xsi:(noNamespace)schemaLocation` attribute, as defined in W3C XML Schema.

Different XML Schema processors are supported:

- By default, the **Java implementation** of XML Schema 1.0 is used (it is based on an old version of Apache Xerces).
- The latest version of Xerces2 provides implementations of XML Schema 1.0 and 1.1. The processor will be applied if you download one of the **binary distributions** and copy the following libraries to the `lib/custom` directory of the full distribution of BaseX:
  - `org.eclipse.wst.xml.xpath2.processor_1.2.0.jar`
  - `cupv10k-runtime.jar`
  - `xercesImpl.jar`
  - `xml-apis.jar`
- **Saxon Enterprise Edition** will be used if you download the **ZIP release** and if you copy `saxon9ee.jar` and a valid license key to the classpath.

**validate:xsd**

**Signatures**

`validate:xsd($input as item()) as empty-sequence(),
validate:xsd($input as item(), $schema as item()) as empty-`
## validation:-xsd

### validate:xsd($input as item(), $schema as item()?, $features as map(*)) as empty-sequence()

**Summary**
Validates the XML $input document against a $schema, using the processor-specific $features.

**Errors**
- error: the validation fails
- init: the validation process cannot be started
- not-found: no XML Schema validator is available.

**Examples**
- Pass on document and schema as nodes:

```xml
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 all documents of a database against the specified schema, using the supplied feature:

```xml
for $city in db:get('cities')
return validate:xsd($city, 'city.xsd',
  map { 'http://javax.xml.XMLConstants/feature/secure-processing':
    true() })
```

### validate:xsd-info

**Signatures**
validate:xsd-info($input as item()) as xs:string*, validate:xsd-info($input as item(), $schema as item()?) as xs:string*, validate:xsd-info($input as item(), $schema as item()?, $features as map(*)) as xs:string*

**Summary**
Validates the XML $input document against a $schema, using the processor-specific $features, and returns warnings, errors and fatal errors in a string sequence.

**Errors**
- init: the validation process cannot be started
- not-found: no XML Schema validator is available.

### validate:xsd-report

**Signatures**
validate:xsd-report($input as item()) as element(report), validate:xsd-report($input as item(), $schema as xs:string?) as element(report), validate:xsd-report($input as item(), $schema as xs:string?, $features as map(*)) as element(report)

**Summary**
Validates the XML $input document against a $schema, using the processor-specific $features, and returns warnings, errors and fatal errors as XML.

**Errors**
- init: the validation process cannot be started
- not-found: no XML Schema validator is available.

### validate:xsd-processor

**Signatures**
validate:xsd-processor() as xs:string

**Summary**
Returns the name of the applied XSD processor.

### validate:xsd-version

**Signatures**
validate:xsd-version() as xs:string

**Summary**
Returns the supported version of XSD Schema.
RelaxNG Validation

Checks whether an XML document validates against a RelaxNG schema. The input document and the 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.

RelaxNG validation will be available if Jing exists in the classpath. The latest version, `jing-20091111.jar`, is included in the full distributions of BaseX. As Jing additionally supports NVDL validation, you can also use the functions to validate the input against NVDL schemas.

validate:rng

**Signatures**

`validate:rng($input as item(), $schema as item()) as empty-sequence()`, `validate:rng($input as item(), $schema as item(), $compact as xs:boolean) as empty-sequence()`

**Summary**

Validates the XML `$input` document against a `$schema`, using the XML or `$compact` notation.

**Errors**

`error`: the validation fails.`init`: the validation process cannot be started.`not-found`: the RelaxNG validator is not available.

**Examples**

• `validate:rng('doc.xml', 'doc.rng')` validates the document `doc.xml` against the specified schema `doc.rng`.

validate:rng-info

**Signatures**

`validate:rng-info($input as item(), $schema as item()) as xs:string*`, `validate:rng-info($input as item(), $schema as item(), $compact as xs:boolean) as xs:string*`

**Summary**

Validates the XML `$input` document against a `$schema`, using the XML or `$compact` notation, and returns warnings, errors and fatal errors in a string sequence.

**Errors**

`init`: the validation process cannot be started.`not-found`: the RelaxNG validator is not available.

validate:rng-report

**Signatures**

`validate:rng-report($input as item(), $schema as xs:string) as element(report)`, `validate:rng-report($input as item(), $schema as xs:string, $compact as xs:boolean) as element(report)`

**Summary**

Validates the XML `$input` document against a `$schema`, using the XML or `$compact` notation, and returns warnings, errors and fatal errors as XML.

**Errors**

`init`: the validation process cannot be started.`not-found`: The RelaxNG validator is not available.

Schematron Validation

If you want to use Schematron for validating documents, install Vincent Lizzi’s excellent Schematron XQuery Module for BaseX:

`repo:install('https://github.com/Schematron/schematron-basex/raw/master/dist/schematron-basex-1.2.xar')`

The following query illustrates how documents are validated. It is directly taken from the GitHub project:
import module namespace schematron = "http://github.com/Schematron/schematron-basex";

let $sch := schematron:compile(doc('rules.sch'))
let $svrl := schematron:validate(doc('document.xml'), $sch)
return (
    schematron:is-valid($svrl),
    for $message in schematron:messages($svrl)
    return concat(schematron:message-level($message), ': ', schematron:message-description(schematron:message-level($message)))
)

## Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>The document cannot be validated against the specified schema.</td>
</tr>
<tr>
<td>init</td>
<td>The validation cannot be started.</td>
</tr>
<tr>
<td>not-found</td>
<td>No validator is available.</td>
</tr>
</tbody>
</table>

## Changelog

**Version 9.2**

- Added: validate:xsd-processor, validate:xsd-version

- Updated: validate:xsd, validate:xsd-info, validate:xsd-report: version argument was dropped (the latest version will always be used)

**Version 9.0**

- Updated: error codes updated; errors now use the module namespace

**Version 8.5**

- Updated: Relative URIs will always be resolved against the static base URI of the query

**Version 8.3**

- Added: validate:rng, validate:rng-info


**Version 7.6**

- Added: validate:xsd-info, validate:dtd-info

The module was introduced with Version 7.3.
Chapter 72. Web Module

This XQuery Module provides convenience functions for building web applications with RESTXQ.

Conventions

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

Functions

web:content-type

<table>
<thead>
<tr>
<th>Signatures</th>
<th>web:content-type($path as xs:string) as xs:string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the content type of a path by analyzing its file suffix. application/octet-stream is returned if the file suffix is unknown.</td>
</tr>
</tbody>
</table>
| Examples     |  *

web:create-url

<table>
<thead>
<tr>
<th>Signatures</th>
<th>web:create-url($url as xs:string, $parameters as map(<em>)) as xs:string, web:create-url($url as xs:string, $parameters as map(</em>), $anchor as xs:string) as xs:string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Creates a new URL from the specified $url string, query string $parameters and an optional $anchor reference. The keys 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.</td>
</tr>
</tbody>
</table>
| Examples     |  *

web:encode-url

<table>
<thead>
<tr>
<th>Signatures</th>
<th>web:encode-url($string as xs:string) as xs:string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>
| Examples     |  *

web:decode-url

<table>
<thead>
<tr>
<th>Signatures</th>
<th>web:decode-url($string as xs:string) as xs:string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Decodes a URL to the original string. Percent-encoded characters are decoded to their UTF8 codepoints, and + characters are rewritten to spaces.</td>
</tr>
</tbody>
</table>
| Examples     |  *

Errors

invalid: the string contains invalid XML characters.
### web:forward

**Signatures**

- `web:forward($path as xs:string) as element(rest:forward)`
- `web:forward($path as xs:string, $parameters as map(*)) as element(rest:forward)`

**Summary**

Creates a server-side RESTXQ forward request to the specified `$path`:

- The client will not get notified of this forwarding.
- Supplied query parameters will be attached to parameters of the current request.
- The `$parameter` argument is processed as described in `web:create-url`.

**Examples**

The function call `web:forward('/a/b')` creates the following result (which will be interpreted as forwarding if RESTXQ is used):

```xml
<rest:forward>/a/b</rest:forward>
```

### web:redirect

**Signatures**

- `web:redirect($url as xs:string) as element(rest:response)`
- `web:redirect($url as xs:string, $parameters as map(*)) as element(rest:response)`
- `web:redirect($url as xs:string, $parameters as map(*), $anchor as xs:string) as element(rest:response)`

**Summary**

Creates a RESTXQ redirection to the specified `$url`. The returned response will only work if no other items are returned by the RESTXQ function. The `$parameters` and `$anchor` arguments are processed as described in (see `web:create-url`).

**Examples**

The query `web:redirect('/a/b')` returns the following result (which will be interpreted as redirection if RESTXQ is used):

```xml
<rest:response xmlns:rest="http://exquery.org/ns/restxq">
    <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($output as map(*)?) as element(rest:response)`
- `web:response-header($output as map(*)?, $headers as map(*)) as element(rest:response)`
- `web:response-header($output as map(*)?, $headers as map(*), $atts as map(*)) as element(rest:response)`

**Summary**

Creates a RESTXQ response header. Serialization parameters and header values can be supplied via the `$output` and `$headers` arguments, and status and message attributes can be attached to the HTTP response element with the `$atts` argument.

- `media-type: application/octet-stream`

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`

**Examples**

- The function call `web:response-header()` returns:

```xml
<rest:response xmlns:rest="http://exquery.org/ns/restxq">
</rest:response>
```
• The following expression returns a media-type for binary data, a caching directive, and the OK status:

```xml
web:response-header(
    map { 'media-type': 'application/octet-stream' },
    map { 'Cache-Control': 'max-age=3600,public' },
    map { 'status': 200, 'message': 'OK' }
)
```

• The following RESTXQ function returns the contents of a file to the client with correct media type:

```xml
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)
    )
};
```

### web:error

**Signatures**

`web:error($status as xs:integer, $message as xs:string) as none,`

**Summary**

Raises an error with the QName `rest:error`, the specified `$message` and the specified `$status` as error value. Calls to this function are equivalent to `fn:error(xs:QName('rest:error'), $message, $status)`. See RESTXQ: Raise Errors to learn how the function is helpful in web applications.

**Examples**

- `web:error(404, "The requested resource cannot be found.")`

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid</td>
<td>A string contains invalid XML characters.</td>
</tr>
<tr>
<td>status</td>
<td>The supplied status code is invalid.</td>
</tr>
</tbody>
</table>

### Changelog

**Version 9.3**

- Added: `web:error, web:forward`

**Version 9.2**

- Updated: `web:create-url, web:redirect: third argument added`.

**Version 9.0**

- Updated: `web:response-header: third argument added; default parameters removed`.
- Updated: error codes updated; errors now use the module namespace

**Version 8.4**

- Updated: `web:response-header: serialization method raw was removed (now obsolete).`
Version 8.2


The module was introduced with Version 8.1.
Chapter 73. WebSocket Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for accessing specific WebSocket functions. This module is mainly useful in the context of WebSockets.

Conventions

- The module will be available if the basex-api library is found in the classpath. This is the case if you use one of the complete distributions of BaseX (zip, exe, war).

- All functions and errors are assigned to the http://basex.org/modules/ws namespace, which is statically bound to the ws prefix.

- As sessions are side-effecting operations, all functions are flagged as non-deterministic. As a result, some query optimizations will be suppressed.

General Functions

**ws:id**

**Signatures**

ws:id() as xs:string

**Summary**

Returns the ID of the current WebSocket.

**Errors**

not-found: No WebSocket with the specified id exists.

**ws:ids**

**Signatures**

ws:ids() as xs:string*

**Summary**

Returns the ids of all currently registered WebSockets.

**ws:path**

**Signatures**

ws:path($id as xs:string) as xs:string

**Summary**

Returns the path of the WebSocket with the specified $id.

**Errors**

not-found: No WebSocket with the specified id exists.

**ws:close**

**Signatures**

ws:close($id as xs:string) as empty-sequence()

**Summary**

Closes the connection of the WebSocket with the specified $id.

**Errors**

not-found: No WebSocket with the specified id exists.

Sending Data

**ws:send**

**Signatures**

ws:send($message as item(), $ids as xs:string*) as empty-sequence()

**Summary**

Sends a $message to the clients with the specified $ids. Ids that cannot be assigned to clients will be ignored. The message will be handled as follows:

- Items of type xs:base64Binary and xs:hexBinary will be transmitted as binary messages.
WebSocket Module

- Function items (maps, arrays) will be serialized as JSON and transmitted as string messages.
- All other items will be serialized with the default serialization options and transmitted as string messages.

**ws:broadcast**

**Signatures**

```xml
ws:broadcast($message as xs:anyAtomicType) as empty-sequence()
```

**Summary**

Broadcasts a `$message` to all connected clients except to the caller. Invocations of this convenience function are equivalent to `ws:send($message, ws:ids()[. ! = ws:id()])`. See `ws:send` for more details on the message handling.

**ws:emit**

**Signatures**

```xml
ws:emit($message as xs:anyAtomicType) as empty-sequence()
```

**Summary**

Emits a `$message` to all connected clients. Invocations of this function are equivalent to `ws:send($message, ws:ids())`. See `ws:send` for more details on the message handling.

**ws:eval**

**Signatures**

```xml
ws:eval($query as xs:anyAtomicType) as xs:string,
ws:eval($query as xs:anyAtomicType, $bindings as map(*)?) as xs:string,
ws:eval($query as xs:anyAtomicType, $bindings as map(*)?, $options as map(*)?) as xs:string
```

**Summary**

Schedules the evaluation of the supplied `$query` and returns the result to the calling WebSocket client. The query can be a URI or a string, and variables and context items can be declared via `$bindings` (see xquery:eval for more details). The following `$options` can be supplied:

- `base-uri`: sets the `base-uri` property for the query. This URI will be used when resolving relative URIs, such as with `fn:doc`.
- `id`: sets a custom job id. The id must not start with the standard `job` prefix, and it can only be assigned if no job with the same name exists.

Query scheduling is recommendable if the immediate query execution might be too time consuming and lead to a timeout.

**Errors**

- `overflow`: Query execution is rejected, because too many jobs are queued or being executed.
- `id`: The specified id is invalid or has already been assigned.

**Examples**

- Schedule a second query that will notify the client 10 seconds later that a message was processed:

  ```xml
  declare
  %ws:message('/tasks', '{$message}')
  function local:message($message) {
    ws:eval('prof:sleep(10000), "Your message has been processed."')
  }
  ```

**WebSocket Attributes**

**ws:get**

**Signatures**

```xml
ws:get($id as xs:string, $name as xs:string) as item()*, ws:get($id as xs:string, $name as xs:string, $default as item()*) as item()*
```

**Summary**

Returns the value of an attribute with the specified `$name` from the WebSocket with the specified `$id`. If the attribute is unknown, an empty sequence or the optionally specified `$default` value will be returned instead.
WebSocket Module

Errors | not-found: No WebSocket with the specified id exists.

**ws:set**

Signatures | ws:set($id as xs:string, $name as xs:string, $value as item()*) as empty-sequence()

Summary | Returns the specified value of the attribute with the specified $name from the WebSocket with the specified $id.

Errors | not-found: No WebSocket with the specified id exists.

**ws:delete**

Signatures | ws:delete($id as xs:string, $name as xs:string) as empty-sequence()

Summary | Deletes an attribute with the specified $name from the WebSocket with the specified $id.

Errors | not-found: No WebSocket with the specified id exists.

**Examples**

**Example 1**

```xml
import module namespace ws = "http://basex.org/modules/ws";

declare %ws:connect('/')
function local:connect() as empty-sequence() {
    let $id := ws:id()
    let $message := json:serialize(map {
        'type': 'Connect',
        'id': $id
    })
    return ws:broadcast($message)
}
```

Explanation:

- The function has a %ws:connect annotation. It gets called if a client successfully creates a WebSocket connection to the path / (check out WebSockets for further information).
- A JSON response is generated, which contains the new client id and a Connect string.
- This response will be sent to all other connected clients.

**Example 2**

```xml
import module namespace ws = "http://basex.org/modules/ws";

declare %ws:message('/', '{$message}')
function local:message($message as xs:string) as empty-sequence() {
    let $message := json:serialize(map { 'message': $message })
    return ws:emit($message)
}
```

Explanation:

- The function has a %ws:message annotation. It gets called if a client sends a new message.
WebSocket Module

- A JSON response is generated, which contains the message string.
- This response will be sent to all connected clients (including the calling client).

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>not-found</td>
<td>No WebSocket with the specified id exists.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.2
- Added: `ws:eval`

This module was introduced with Version 9.1.
Chapter 74. XQuery Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for parsing and evaluating XQuery strings at runtime, and to run code in parallel.

Conventions

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

Evaluation

xquery:eval

Signatures

xquery:eval($query as xs:anyAtomicType) as item()*,
xquery:eval($query as xs:anyAtomicType, $bindings as map(*)?) as item()*,
xquery:eval($query as xs:anyAtomicType, $bindings as map(*)?, $options as map(*)?) as item()*,

Summary

Evaluates the supplied $query and returns the resulting items. If the query is of type xs:anyURI, the module located at this URI will be retrieved (a relative URI will be resolved against the static base URI). Otherwise, the input is expected to be of type xs:string. 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:

- 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. Moreover, please note that this option enforces garbage collection, so it will take some additional time, and it requires GC to be enabled in your JVM.
- base-uri: set base-uri property for the query. Overwrites the base URI of the query; will be used when resolving relative URIs by functions such as fn:doc.
- pass: passes on the original error info (line and column number, optional file uri). By default, this option is false.

Errors

update: the query contains updating expressions, permission: insufficient permissions for evaluating the query, timeout: query execution exceeded timeout, limit: query execution exceeded memory limit, nested: nested query evaluation is not allowed. Any other error that may occur while evaluating the query.

Examples

- xquery:eval("1+3") returns 4.
- If a URI is supplied, the query in the specified file will be evaluated:

xquery:eval(xs:anyURI('cleanup.xq'))
You can bind the context and e.g. operate on a certain database only:

```xquery
xquery:eval("//country", map { '': db:get('factbook') })
```

The following expressions use strings as keys. All of them return 'XML':

```xquery
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':

```xquery
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:eval-update

**Signatures**

- `xquery:eval-update($query as xs:anyAtomicType) as item()*
- `xquery:eval-update($query as xs:anyAtomicType, $bindings as map(*)?) as item()*
- `xquery:eval-update($query as xs:anyAtomicType, $bindings as map(*)?, $options as map(*)?) as item()*

**Summary**

Evaluates a query as updating expression. All updates will be added to the Pending Update List of the main query and performed after the evaluation of the main query. The rules for all arguments are the same as for `xquery:eval`.

**Errors**

- `update`: the query contains no updating expressions.
- `permission`: insufficient permissions for evaluating the query.
- `timeout`: query execution exceeded timeout.
- `limit`: query execution exceeded memory limit.
- `nested`: nested query evaluation is not allowed.
- Any other error that may occur while evaluating the query.

**Examples**

- Removes entries from a temporary databases and returns an info string:

```xquery
xquery:eval-update("delete node db:get('tmp')/*,
    update:output('TEMPORARY DATABASE WAS CLEANED UP')"
```

### Parsing

**xquery:parse**

*Updated with Version 10:* $query can additionally be of type `xs:anyURI`.

**Signatures**

- `xquery:parse($query as xs:anyAtomicType) as item()?
- `xquery:parse($query as xs:anyAtomicType, $options as map(*)?) as item()?`
Summary
Parses the specified $query as XQuery module and returns the resulting query plan. If the query is of type xs:anyURI, the module located at this URI will be retrieved (a relative URI will be resolved against the static base URI). Otherwise, the input is expected to be of type xs:string. The $options parameter influences the output:

- **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. The naming of the expressions in the query plan may change over time.
- **pass**: by default, the option is false. If an error is raised, the line/column number and the optional file uri will refer to the location of the function call. If the option is enabled, the line/column and file uri will be adopted from the raised error.
- **base-uri**: set base-uri property for the query. This URI will be used when resolving relative URIs by functions such as fn:doc.

Errors
Any error that may occur while parsing the query.

Examples
- xquery:parse("1 + 3") returns:

```xml
<MainModule updating="false">
  <QueryPlan compiled="false">
    <Arith op="+">
      <Int value="1" type="xs:integer"/>
      <Int value="3" type="xs:integer"/>
    </Arith>
  </QueryPlan>
</MainModule>
```

Parallelized Execution
Parallel query execution is recommendable if you have various calls that require a lot of time, but that cannot be sped up by rewriting the code. This is e.g. the case if external URLs are called. If you are parallelizing local data reads (such as the access to a database), single-threaded queries will usually be faster, because parallelized access to disk data often results in randomized access patterns, which will rarely be optimized by the caching strategies of HDDs, SSDs, or the operating system.

**xquery:fork-join**

**Signatures**
xquery:fork-join($functions as function(*)*) as item()*

**Summary**
This function executes the supplied (non-updating) functions in parallel.

**Examples**
- The following function sleeps in parallel; it will be finished in 1 second if your system has at least 2 cores:

```xquery
let $f := function() { prof:sleep(1000) }
return xquery:fork-join(($f, $f))
```
- In the following query, up to four URLs will be requested in parallel:

```xquery
xquery:fork-join{
  for $segment in 1 to 4
  let $url := 'http://url.com/path/' || $segment
  return function() { http:send-request((), $url) }
}
```

**Errors**
error: an unexpected error occurred.
permission  Insufficient permissions for evaluating the query.
update   updating expression found or expected.
timeout  Query execution exceeded timeout.
memory   Query execution exceeded memory limit.
nested   Nested query evaluation is not allowed.
error    An unexpected error occurred.

Changelog

Version 10

• Deleted: xquery:parse-uri (merged with xquery:parse)

• Updated: xquery:parse: Template:$query can additionally be of type xs:anyURI.

Version 9.2


Version 9.0

• Added: xquery:invoke-update

• Updated: xquery:eval: pass option added

• Updated: xquery:parse, xquery:parse-uri:base-uri option added

• Updated: xquery:update renamed to xquery:eval-update

• Updated: error codes updated; errors now use the module namespace

Version 8.5

• Added: xquery:fork-join

• Updated: xquery:eval:base-uri option added

• Updated: Relative URIs will always be resolved against the static base URI of the query

• Deleted: xquery:type (moved to Profiling Module)

Version 8.4

• Added: xquery:parse-uri

• Updated: xquery:parse: pass option added

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
XQuery Module

• 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 75. XSLT Module

This XQuery Module contains functions and variables to perform XSL transformations.

By default, this module uses Java’s XSLT 1.0 Xalan implementation to transform documents. XSLT 3.0 will be enabled if Version 9 or 10 of the Saxon XSLT Processor is found in the class path (see Distributions for more details). A custom transformer can be specified by assigning a class to the system property `javax.xml.transform.TransformerFactory`, e.g. directly in Java:

```java
System.setProperty(
    "javax.xml.transform.TransformerFactory",
    "org.custom.xslt.TransformerFactoryImpl");
```

```java
Context ctx = new Context();
String result = new XQuery("xslt:transform('...', '...')").execute(ctx);
...
ctx.close();
```

Conventions

All functions and errors in this module are assigned to the `http://basex.org/modules/xslt` namespace, which is statically bound to the `xslt` 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 "3.0"). "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(), $params as map(*)) as node(),
xslt:transform($input as item(), $stylesheet as item(), $args as map(*), $options as map(*)) 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 stylesheet URI,
- `xs:string`, containing the document in its string representation, or
- `node()`, containing the document itself.

XML Catalog files will be considered when resolving URIs. Variables can be bound to a stylesheet via `$args` (only strings are supported when using XSLT 3.0 and Saxon). The following `$options` are available:
Error

error: 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 map(*)), $options as map(*)?) as xs:string

Summary

Transforms the document specified by $input, using the XSLT template specified by $stylesheet, and returns the result as string. The semantics of $params and $options is the same as for xslt:transform.

Error

error: an error occurred during the transformation process.

xslt:transform-report

Signatures

xslt:transform-report($input as item(), $stylesheet as item()) as xs:string,
xslt:transform-report($input as item(), $stylesheet as item(), $params as map(*)), $options as map(*)?) as xs:string

Summary

Transforms the document specified by $input, using the XSLT template specified by $stylesheet, and returns a map with the following keys:

- result: The transformation result: One or more document nodes, or (if the result cannot be converted to XML) an item of type xs:untypedAtomic.
- messages: Informational output generated by xsl:message elements: A sequence of arrays. The arrays consist of XML elements, or (for those messages that cannot be converted to XML) items of type xs:untypedAtomic.

The semantics of $params and $options is the same as for xslt:transform. For the moment, messages can only be returned with recent versions of Saxon.

- error (optional): An error string, which would be raised as an error by the other functions of this module.

Examples

Example 1: XSL transformation, with XML and XSL supplied as nodes

Query:

!(: Outputs the result as html. :)!
declare option output:method 'html';

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>
</books>
Example 2: Textual XSL transformation

Query:

```
xslt:transform-text(<dummy/>, 'basic.xslt')
```

**Result:**

```
123
```

Example 3: XSL transformation with variable assignment

Query:

```
let $in := <dummy/>
let $style := doc('variable.xsl')
return xslt:transform($in, $style, map { "v": 1 })
```

**Result:**

```

```

```
Example 4: XSL transformation, yielding a result and info messages

Query:

```xml
<xsl:transform version='2.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:message><msg>START...</msg></xsl:message>
  <xml>123</xml>
  <xsl:message select='4, 5, "...END"'/>
</xsl:template>
</xsl:transform>
```

Result:

```json
map {
  "messages": (['<msg>START...</msg>'], ['4 5 ...END']),
  "result": <xml>123</xml>
}
```

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>An error occurred during the transformation process.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.7

- Added: `xslt:transform-report`

Version 9.2

- Updated: Support for XML Catalog files added.

Version 9.0

- Updated: `xslt:transform`, `xslt:transform-text`: `$options` argument added.
- Updated: error codes updated; errors now use the module namespace

Version 7.6

- Added: `xslt:transform-text`

Version 7.3

- Updated: `$xslt:processor` → `xslt:processor`, `$xslt:version` → `xslt:version`
Part VIII. Developing
Chapter 76. 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!

Web Technology
• RESTXQ: Write web services with XQuery
• REST: Access and update databases via HTTP requests
• WebDAV: Access databases from your filesystem

APIs
• Clients: Communicate with BaseX using C#, PHP, Python, Perl, C, ...
• Java Examples: Code examples for developing with BaseX
• XQJ API: Closed source, implemented by Charles Foster (restricted to XQuery 3.0)
• XQuery for Scala API, based on XQJ and written by Dino Fancellu

Extensions
• Service/daemon: Install BaseX server as a service
• Android: Running BaseX with Android

Code, Questions, Bugs
• The Source Code is available on GitHub.
• For questions, bug reports and feature requests, please write to our mailing list
• The Issue Tracker contains confirmed bugs and feature requests.
Chapter 77. 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 plugins we use are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update URL</th>
<th>Eclipse Marketplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpotBugs</td>
<td>Analyze project at byte code level</td>
<td><a href="https://spotbugs.github.io/eclipse/">https://spotbugs.github.io/eclipse/</a></td>
<td>install</td>
</tr>
<tr>
<td>UCDetector</td>
<td>Unnecessary code detector</td>
<td><a href="http://ucdetector.sourceforge.net/">http://ucdetector.sourceforge.net/</a></td>
<td>update</td>
</tr>
</tbody>
</table>

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 subdirectories:

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 contains errors or warnings, you may need to switch to Java 11 (Windows → Preferences → Java → Installed JREs).

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.
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

Using Git & Eclipse

Clone

- In the Package Explorer to the left, use right-click and choose Import...
- Select Projects from Git and click Next
- Choose the Clone option 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".
Projects from Git

Clone
GitHub URI

Select Branch
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 check out and click finish

- You are now ready to contribute.
Select a Git Repository

Import existing Projects
EGit & SSH

The Eclipse git plugin uses the JSch library, which had problems with RSA SSH keys in Linux and possibly other platforms. If the problem persists, the path to the native SSH executable can be assigned to the GIT_SSH variable.

Using Git on Command-Line

Note: this is not intended to be a complete git reference; its 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 username 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 Repository

$ git clone git@github.com:$username/$project.git
Cloning into $project...
Enter passphrase for key '/home/user/.ssh/id_rsa':
...
$ ls -d -l $PWD/*
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 SEDITOR 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 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 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**

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-feature
Counting 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-feature
```

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.

**Links**

- GitHub: [git Installation Guide](http://git-scm.com)
- Comprehensive Getting Starting Guide on [GitHub](https://help.github.com)
- The [git book](https://git-scm.com/book)
Chapter 79. 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:

```xml
<repositories>
  <repository>
    <id>basex</id>
    <name>BaseX Maven Repository</name>
    <url>http://files.basex.org/maven</url>
  </repository>
</repositories>

<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex</artifactId>
  <version>7.6</version>
</dependency>
```

BaseX Main Package

```xml
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
</dependency>
```

APIs and Services

...including APIs and the REST,.RESTXQ and WebDAV services:

```xml
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
</dependency>
```
XQJ API

The XQJ API is hosted at http://xqj.net:

```xml
<repository>
    <id>xqj</id>
    <name>XQJ Maven Repository</name>
    <url>http://xqj.net/maven</url>
</repository>
```

...
Chapter 80. 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:

https://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):

https://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:

https://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/basex/

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:

https://files.basex.org/maven/org/basex/

**Linux**

BaseX can also be found in some Linux distributions, such as Debian, Ubuntu and archlinux:

- Debian: https://packages.debian.org/sid/basex
- Ubuntu: https://launchpad.net/ubuntu/+source/basex
- Arch Linux: https://aur.archlinux.org/packages/basex
Chapter 81. 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.

If new strings are added to BaseX, they will automatically be added to your language files in English. The history view in GitHub is helpful to see which strings have recently been updated to a file.

**Updating BaseX.jar**

You can 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 into 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 can also directly assign a language in the .basex configuration file, which is placed in your home directory. The language is assigned to the LANG option. In order to see where the text keys are used within BaseX, you can set LANGKEY to true.
Part IX. Web Technology
Chapter 82. RESTXQ

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. It has been inspired by the Java JAX-RS API: It provides 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 original 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
- %input annotations, support for input-specific content-type parameters
- %rest:single annotation to cancel running RESTXQ functions
- Quality factors in the Accept header will be evaluated
- Support for server-side quality factors in the %rest:produces annotation
- Better support for the OPTIONS and HEAD methods

Introduction

Preliminaries

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.

If a RESTXQ URL is requested, the RESTXQPATH module directory and its subdirectories will be traversed, and all XQuery files will be parsed for functions with RESTXQ annotations. Subdirectories that include an .ignore file will be skipped.

To speed up processing, the functions of the existing XQuery modules are automatically cached in main memory:

- Functions will be invalidated and parsed again if the timestamp of their module changes.
- File monitoring can be adjusted via the PARSERESTXQ option. In productive environments with a high load, it may be recommendable to change the timeout, or completely disable monitoring.
- If files are replaced while the web server is running, the RESTXQ module cache should be explicitly invalidated by calling the static root path /.init or by calling the rest:init function.
Examples

A first RESTXQ function is shown below:

```xquery
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:

```xml
<response>
    <title>Hello World!</title>
</response>
```

The next function demonstrates a POST request:

```xquery
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/`)...

```bash
curl -i -X POST --data "message='CONTENT'" http://localhost:8984/form
```

...you will receive something similar to the following result:

```plaintext
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:

```xml
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($code) { ... };
```

```
(: Matches all other all paths starting with "app/" :) 
declare %rest:path("app/{\$path=.+}\") 
function page:others($path) { ... };
```

If multiple path path candidates are found for the request, the one with more segments will be preferred.

### Content Negotiation

Functions can be restricted to specific Media Types. The default type is */*. Multiple types can either be specified by a single or by multiple annotations.

#### Consuming Data

A function will only be taken into consideration if the HTTP Content-Type header of the request matches one of the given types:

```xml
declare
%rest:POST("{$body}\")
%rest:path("/xml")
%rest:consumes("application/xml")
%rest:consumes("text/xml")
function page:xml($body) { $body };
```

#### Producing Data

A function will only be chosen if the HTTP Accept header of the request matches one of the given types:

```xml
declare
%rest:path("/xml")
%rest:produces("application/xml", "text/xml")
function page:xml() { <xml/> };
```

Note that the annotations will not affect the type of the actual response: You will need to supply an additional %output:media-type annotation or (if a single function may produce results of different types) generate an apt Custom Response.

### Quality Factors

A client can supply quality factors to influence the server-side function selection process. If a client sends the following HTTP header with quality factors...

```xml
Accept: */*;q=0.5,text/html;q=1.0
```

...and if two RESTXQ functions exist for the addressed path with two different annotations for producing data...

```xml
declare function %rest:produces("text/html") ...
```
... declare function %rest:produces("*/") ...

...the first of these function will be chosen, as the quality factor for text/html documents is highest.

As we cannot ensure that the client may supply quality factors, the selection process can also be controlled server-side. The qs parameter can be attached server-side to the Media Type. If multiple functions are left in the selection process, the one with the highest quality factor will be favored:

```
declare function %rest:produces("application/json;qs=1") ...
... declare function %rest:produces("*/;qs=0.5") ...
```

**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. An optional body variable can be supplied as second argument:

```
declare %rest:path("binary-size")
  %rest:method("SIZE", "{$body}")
  function page:patch($body as xs:base64Binary)
    ( "$Request method: " || request:method(),
      "Size of body: " || bin:length($body)
    ) ;
```

If an OPTIONS request is received, and if no function is defined, an automatic response will be generated, which includes an Allow header with all supported methods.

If a HEAD request is received, and if no function is defined, the corresponding GET function will be processed, but the response body will be discarded.

**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:

<table>
<thead>
<tr>
<th>Content-Type</th>
<th>Parameters (; name=value)</th>
<th>Type of resulting XQuery item</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/xml, application/xml</td>
<td></td>
<td>document-node()</td>
</tr>
<tr>
<td>text/*</td>
<td></td>
<td>xs:string</td>
</tr>
<tr>
<td>application/json</td>
<td>JSON Options</td>
<td>document-node() or map(*)</td>
</tr>
</tbody>
</table>

401
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 with the input prefix. The following function interprets the input as text with the CP1252 encoding and treats the first line as header:

```xml
declare
%rest:path("/store.csv")
%rest:POST("{$csv}";
%input:csv("header=true,encoding=CP1252")
function page:store-csv($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:

```xml
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):

```xml
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="{$add}"/>
};
```

### HTML Form Fields

Form parameters are specified the same way as query parameters:

```xml
%rest:form-param("city", "{$city}"; "no-city-specified")
```
The values are the result of HTML forms submitted with the (default) content type `application/x-www-form-urlencoded`:

```html
<form action="/process" method="POST" enctype="application/x-www-form-urlencoded">
  <input type="text" name="city"/>
  <input type="submit"/>
</form>
```

**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):

```html
<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:

```restxq
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)}"></file>
  )
}
```

**HTTP Headers**

Header parameters are specified the same way as query parameters:

```restxq
%rest:header-param("User-Agent", "{$user-agent}")
%rest:header-param("Referer", "{$referer}"")
```

**Cookies**

Cookie parameters are specified the same way as query parameters:

```restxq
%rest:cookie-param("username", "{$user}")
%rest:cookie-param("authentication", "{$auth}")
```

**Query Execution**

In many web search scenarios, user input from browser forms is processed and search results are returned. Such operations can be made more interactive by sending a new search request to the server with each key click. However, this may lead to many parallel server-side requests, from which only the result of the last request will be relevant for the client.

With the `%rest:single` annotation, it can be enforced that only one instance of a function will run at the same time and for the same client. If the same function will be called for the second time, a currently executed query will be stopped, and the HTTP error code 460 will be returned instead:

```restxq
(: If fast enough, returns the result. Otherwise, if called again, raises 460 :) declare %rest:path("/search") %rest:query-param("term", "{$term}")
```
By adding a string value to with the annotation, functions can be bundled together, and a running query can be canceled by calling another one that has the same annotation value. This is shown by another example, in which the first function can be interrupted by the second one. If you call both functions in separate browser tabs, you will note that the first tab will return 460, and the second one will return <xml>stopped</xml>.

The following things should be noted:

• If a query will be canceled, there will be no undesirable side effects. For example, it won’t be possible to abort a query if it is currently updating the database or performing any other I/O operations. As a result, the termination of a running query can take some more time as expected.

• The currently executed function is bound to the current session. This way, a client will not be able to cancel requests from other clients. As a result, functions can only be stopped if there was at least one previous successful response, in which initial session data was returned to the client.

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”).

A Server-Timing HTTP header is attached to each response. It indicates how much time was spent for parsing, compiling, evaluating and serializing the query. The last value will not necessarily reflect the full time for serializing the result, as the header is generated before the result is sent to the client. Server-side serialization can be enforced by annotating a function with the %rest:single annotation.

Custom Response

Custom responses can be generated in XQuery by returning an rest:response element, an http:response child node that matches the syntax of the EXPath HTTP Client Module specification, and optional child nodes that will be serialized as usual. A function that yields a response on an unknown resource may look as follows:

declare %output:method("text") %rest:path(""") function page:error404() {
  <rest:response>
    <http:response status="404">
      <http:header name="Content-Language" value="en"/>
      <http:header name="Content-Type" value="text/plain; charset=utf-8"/>
    </http:response>
  </rest:response>,
  "The requested resource is not available."
}
Forwards and Redirects

Redirects

The server can invite the client (e.g., the web browser) to make a second request to another URL by sending a 302 response:

```xml
<rest:response>
  <http:response status="302">
    <http:header name="Location" value="new-location"/>
  </http:response>
</rest:response>
```

The convenience function `web:redirect` can be called to create such a response.

In the XQuery context, redirects are particularly helpful if Updates are performed. An updating request may send a redirect to a second function that generates a success message, or evaluates an updated database:

```query
declare %updating %rest:path('/app/init') function local:create() {
  db:create('app', <root/>, 'root.xml'),
  db:output(web:redirect('/app/ok'))
};
```

```query
declare %rest:path('/app/ok') function local:ok() {
  'Stored documents: ' || count(db:get('app'))
};
```

Forwards

A server-side redirect is called forwarding. It reduces traffic among client and server, and the forwarding will not change the URL seen from the client’s perspective:

```xml
<rest:forward>new-location</rest:forward>
```

The fragment can also be created with the convenience function `web:forward`.

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:

```query
declare option output:media-type 'text/plain';
```

```query
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:

```query
declare %rest:path("cities")
```
The next function, when called, generates XHTML headers, and text/html will be set as content type:

```
declare
%rest:path("done")
%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

If an error is raised when RESTXQ code is parsed, compiled or evaluated, an HTTP response with the status code 500 is generated.

By default, all server-side errors will be passed on to the client. This is particularly helpful during the development process. In a productive environment, however, it is advisable not to expose errors to the client. This can be realized via the RESTXQERRORS option. If disabled,

- XQuery modules that cannot be parsed will be ignored and
- full error messages and stack traces will be suppressed and not included in the HTTP response.

The full error information can still be looked up in the database logs.

## Raise Errors

With `web:error`, you can abort query evaluation, enforce a premature HTTP response and report errors back to the client:

```
declare
  %rest:path("/teapot")
function page:teapot() {
  web:error(418, "I'm a pretty teapot")
}
```

In contrast to the standard `fn:error` function, a status code can be supplied, and the response body will only contain the specified error message and no stack trace.
Catch 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 try/catch construct:

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>prefix:name,Q{uri}name</td>
<td>err:FORG0001, Q{<a href="http://www.w3.org/2005/xqt-errors%7DFORG0001">http://www.w3.org/2005/xqt-errors}FORG0001</a></td>
</tr>
<tr>
<td>2</td>
<td>prefix:* , Q{uri}*</td>
<td>err:* , Q{<a href="http://www.w3.org/2005/xqt-errors%7D">http://www.w3.org/2005/xqt-errors}</a>*</td>
</tr>
<tr>
<td>3</td>
<td>* : name</td>
<td>* : FORG0001</td>
</tr>
<tr>
<td>4</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

All error codes that are specified for a function must have the same precedence. The following rules apply when catching errors:

- Codes with a higher precedence (smaller number) will be given preference.
- 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:

```xml
<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:

```xml
declare $rest:path("/error404") function page:error404() {
  "URL: " || request:attribute("javax.servlet.error.request_uri") || ", " || 
  "Error message: " || request:attribute("javax.servlet.error.message")

```

Catch 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):
User Authentication

If you want to provide restricted access to parts of a web applications, you will need to check permissions before returning a response to the client. The Permissions layer is a nice abstraction for defining permission checks.

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 URLs 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:

```xml
import module namespace request = "http://exquery.org/ns/request";

declare %rest:path("/host-name") function page:host() {
  'Remote host name: ' || request:remote-hostname()
};
```

References

Documentation:
- RESTXQ Specification, First Draft
- RESTful XQuery, Standardised XQuery 3.0 Annotations for REST. Paper, XMLPrague, 2012
- Web Application Development. Slides from XMLPrague 2013

Examples:
- Sample code combining XQuery and JavaScript: Materials and paper from Amanda Galtman, Balisage 2016.
- DBA: The Database Administration interface, bundled with the full distributions of BaseX.

Changelog

Version 9.6
- Updated: Response: Server-Timing HTTP header.

Version 9.5
- Updated: Raise Errors: Status code 400 changed to 500, omit stack trace.

Version 9.3
- Updated: Custom Methods: Better support for the OPTIONS and HEAD methods.
- Updated: XQuery Errors: Suppress stack trace and error code in the HTTP response.
- Removed: rest:redirect element (web:redirect can be used instead)

Version 9.2
- Updated: Ignore XQuery modules that cannot be parsed
Version 9.0
• Added: Support for server-side quality factors in the %rest:produces annotation
• Updated: Status code 410 was replaced with 460
• Removed: restxq prefix
Version 8.4
• Added: %rest:single annotation
Version 8.1
• Added: support for input-specific content-type parameters
• Added: %input annotations
Version 8.0
• Added: Support for regular expressions 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 83. Permissions

Read this entry online in the BaseX Wiki.

This page presents the web application permission layer of BaseX, which can be used along with RESTXQ.

Non-trivial web applications require a user management: Users need to log in to a web site in order to get access to protected pages; Depending on their status (role, user group, ...), they can be offered different views; etc. The light-weight permission layer simplifies permission checks a lot:

- Permission strings can be attached to RESTXQ functions.
- With security functions, you can ensure that access to RESTXQ functions will only be granted to clients with sufficient permissions.

Preliminaries

All permission annotations are assigned to the http://basex.org/modules/perm namespace, which is statically bound to the perm prefix.

Annotations

Permission Strings

With the %perm:allow annotation, one or more permission strings can be attached to a RESTXQ function:

```
!(: Login page (visible to everyone). :) declare
  %rest:path("/") %output:method("html")
  function local:login() {
    <html>
      Please log in:
      <form action="/login-check" method="post">
        <input name="name"/>
        <input type="password" name="pass"/>
        <input type="submit"/>
      </form>
    </html>
  };

!(: Main page (restricted to logged in users). :) declare
  %rest:path("/main") %output:method("html")
  function local:main() {
    <html>
      Welcome to the main page:
      <a href="/main/admin">admin area</a>,
      <a href="/logout">log out</a>.
    </html>
  };

!(: Admin page. :) declare
  %rest:path("/main/admin") %output:method("html")
  %perm:allow("admin")
  function local:admin() {
    <html>
```

410
Welcome to the admin page.
</html>
}

The permission strings may denote ids, users, user groups, applications, or any other realms. It is completely up to the user which strings are used, and which functions will be annotated. In the given example code, only the last function has a $perm:allow annotation.

**Checking Permissions**

Functions that are marked with $perm:check are so-called *Security Functions*. These functions will be invoked before the actually requested function will be evaluated. Two arguments can be specified with the annotation:

- A path can be specified as first argument:
  - The security function will only be called if the path of the client request starts with the specified path.
  - In contrast to RESTXQ, all subordinate paths will be accepted as well.
  - If no path argument is specified, / is assigned instead.

- A variable can be specified in the second argument. A map with the following keys will be bound to that variable:
  - $allow: Permission strings attached to the requested function; may be empty.
  - $path: Original path of the client request.
  - $method: Method of the client request (GET, POST,...).
  - $authorization: Value of the HTTP Authorization header string; may be empty.

An example:

```xquery
import module namespace Session = 'http://basex.org/modules/session';

{~
  : Global permission checks.
  : Rejects any usage of the HTTP DELETE method.
  :}
declare $perm:check $rest:DELETE function local:check() {
  error(), 'Access denied to DELETE method.'
};

{~
  : Permission check: Area for logged-in users.
  : Checks if a session id exists for the current user; if not, redirects to the login page.
  :}
declare $perm:check('/main') function local:check-app() {
  let $user := Session:get('id')
  where empty($user)
  return web:redirect('/')
};

{~
  : Permissions: Admin area.
  : Checks if the current user is admin; if not, redirects to the main page.
  : @param $perm  map with permission data
  :}
declare $perm:check('/main/admin', '{$perm}') function local:check-admin($perm) {
  let $user := Session:get('id')
  where not(user:list-details($user)/@permission = $perm?allow)
  return web:redirect('/main')
};
```
Some notes:

- If several permission functions are available that match the user request, all of them will be called one after another. The function with the shortest path will be called first. Accordingly, in the example, if the /main/admin URL is requested, all three security functions will be run in the given order.

- If a security function raises an error or returns any XQuery value (e.g. a redirection to another web page), no other functions will be invoked. This means that the function that has been requested by the client will only be evaluated if all security functions yield no result and no error.

- As shown in the first function, the $perm:check annotation can be combined with other RESTXQ annotations, excluding $rest:path and $rest:error.

- In the example, it is assumed that a logged in user is bound to a session variable (see further below).

The permission layer was designed to provide as much flexibility as possible to the web application developer: It is possible to completely work without permission strings, and realize all access checks based on the request information (path, method, and properties returned by the Request Module). It is also possible (but rather unhandy) to accompany each RESTXQ function by its individual security function. The bare minimum is a single $perm:check function. Without this function, existing $perm:allow annotations will be ignored.

**Authentication**

There are numerous ways how users can be authenticated in a web application (via OAuth, LDAP, ...). The approach demonstrated on this page is pretty basic and straightforward:

- A login HTML page allows you to enter your credentials (username, password).

- A login check function checks if the typed in data matches one of the database users. If the input is valid, a session id will be set, and the user will be redirected to the main page. Otherwise, the redirection points back to the login page.

- A logout page deletes the session id.

The following lines of code complete the image:

```xml
declare
%rest:path("/login-check")
%rest:query-param("name", "{$name}")
%rest:query-param("pass", "{$pass}")
function local:login($name, $pass) {
  try {
    user:check($name, $pass),
    Session:set('id', $name),
    web:redirect("/main")
  } catch user:* {
    web:redirect("/")
  }
};

declare
%rest:path("/logout")
function local:logout() {
  Session:delete('id'),
  web:redirect("/")
};;
```

For a full round trip, check out the source code of the DBA that is bundled with BaseX.

**Changelog**

Version 9.1
• Added: authorization value in permissions map variable

The Module was introduced with Version 9.0.
Chapter 84. WebSockets

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the WebSockets API of BaseX. WebSocket is a communication protocol for providing full-duplex communication: Data can be sent in both directions and simultaneously.

Please note that the current WebSocket implementation relies on Jetty’s WebSocket servlet API. Other web servers may be supported in future versions.

Introduction

Protocol

Use WebSockets if you have to exchange data with a high frequency or if you have to send messages from the server to the client without techniques like polling. In contrast to REST, WebSockets use a single URL for the whole communication.

The WebSocket protocol was standardized in RFC 6455 by the IETF. After an initial HTTP request, all communication takes place over a single TCP connection. Unlike the HTTP protocol, a connection will be kept alive, and a server can send unsolicited data to the client.

For establishing a WebSocket connection, a handshake request is sent by the client. The web server returns a handshake response. If the handshake is successful, the persistent connection will be open until the client or the server closes it, an error occurs or a timeout happens. It is possible to transmit all kind of data, binary or text. The BaseX WebServer handles the handshake completely. You just have to define some limits of the connection in the web.xml and specify functions for WebSocket events like onConnect and onMessage.

Notice that there is no specification of a message protocol. The WebSocket protocol just specifies the message architecture but not how the payload of the messages is formatted. To agree on a format between the server and the client one can use sub-protocols.

Some older browsers don’t support the WebSocket protocol. Therefore you can use fallback options like Ajax. JavaScript client libraries like SockJS can be used for building client applications. The library takes care of how to establish the real-time connection. If the WebSocket protocol isn’t supported, it uses polling. You have to provide server functions for the fallback solutions if you have to support fallbacks.

Preliminaries

There are a bunch of annotations depending to WebSockets for annotating XQuery functions. When a WebSocket message arrives at the server, an XQuery function will be invoked that matches the constraints indicated by its annotations.

If a WebSocket function is requested (like connecting to the path /, sending a message to the path /path,...), the module directory and its subdirectories will be traversed, and all XQuery files will be parsed for functions with WebSocket annotations. Subdirectories that include an .ignore file will be skipped.

To speed up processing, the functions of the existing XQuery modules are automatically cached in main memory. For further information on cache handling, check out the RESTXQ introduction.

Configuration

- The WebSocket servlet can be enabled and disabled in the web.xml configuration file. You can specify further configuration options, such as maxIdleTime, maxTextMessageSize, and maxBinaryMessageSize.

- The default limit for messages is 64 KB. If you a message exceeds the default or the specified limit, an error will be raised and the connection will be closed.
Annotations

To tag functions as WebSocket functions you have to use annotations. The annotation is written after the keyword declare and before the keyword function. For the context of WebSockets there are some annotations listed below. Functions which are annotated with a WebSocket annotation will be called if the appropriate event occurs. For example, the function annotated with ws:connect('/') will be executed if a client establishes a connection with the WebSocket root path (which is, by default, ws/). By using annotations, it’s easy to provide an API for your WebSocket connection. You just have to specify what to do when a WebSocket Event occurs, annotate it with the corresponding annotation and the Servlet will do the rest for you.

ws:connect(path)

Called directly after a successful WebSocket handshake. The path specifies the path which a client is connected to:

```java
declare %ws:connect('/') function local:connect() { }
```

You can specify here how to handle your users, e.g. save a name as a WebSocket attribute. Furthermore, you can check header parameters for validity.

ws:message(path, message)

Called when a client message arrives at the server. The path specifies the path which a client is connected to. The message string contains the name of the variable to which the message will be bound:

```java
declare %ws:message('/', '{$info}') function local:message($info) { }
```

The value will be of type xs:string or xs:base64Binary. As there is no fixed message protocol, the client needs to take care of the message syntax.

ws:error(path, message)

Called when an error occurs. The path specifies the path which a client is connected to. The message string contains the name of the variable to which the message will be bound:

```java
declare %ws:error('/', '{$error}') function local:error($error) { }
```

Usually, errors happen because of bad/malformed incoming packets. The WebSocket connection gets closed after the error handling.

ws:close(path)

Called when the WebSocket closes. The path specifies the path which a client is connected to:

```java
declare %ws:close('/') function local:connect() { }
```

The WebSocket is already closed when this annotation is called so there can be no return.

ws:header-param(name, variable[, default])

For accessing connection-specific properties like the HTTP version. The value will be bound to the specified variable. If the property has no value, an optional default value will be assigned instead:

```java
%ws:close('host', '{$host}')
%ws:header-param('host', '{$host}')
function local:close($host) {
  admin:write-log('Connection was closed: ' || $host)
}
```

The following parameters are available:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**WebSockets**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The host of the request URI.</td>
</tr>
<tr>
<td>http-version</td>
<td>The HTTP version used for the request.</td>
</tr>
<tr>
<td>is-secure</td>
<td>Indicates if the connection is secure.</td>
</tr>
<tr>
<td>origin</td>
<td>The WebSocket origin.</td>
</tr>
<tr>
<td>protocol-version</td>
<td>The version of the used protocol.</td>
</tr>
<tr>
<td>query-string</td>
<td>The query string of the request URI.</td>
</tr>
<tr>
<td>request-uri</td>
<td>The Request URI to use for this request.</td>
</tr>
<tr>
<td>sub-protocols</td>
<td>List of configured sub-protocols.</td>
</tr>
</tbody>
</table>

General information on the request can be retrieved via the Request Module.

**Writing Applications**

The WebSocket Module contains functions for interacting with other clients or manage specific clients. For example, you can store and access client-specific properties for a WebSocket connection or close the connection of clients.

Note that one WebSocket connection can be opened per browser tab. In contrast, only one HTTP session exists for multiple tabs in a browser. If you want to keep client-specific data on the web server, you can either store them in HTTP sessions or in the WebSocket connection.

Note further that the results of functions annotated with `%ws:close` or `%ws:error` will not be transmitted to the client. Both annotations have rather been designed to gracefully close connections, write log data, remove clients from session data, etc.

For keeping the connection alive it is recommendable to use heart-beats, and send regular pings to the server. There is no ideal timespan for sending pings: It should not be sent too often, but you should also consider possible network latencies.

If your HTTP connection is secure, you should use the `wss` instead of the `ws` scheme.

If you get the `[basex:ws] WebSocket connection required error`, you may be attempting to call WebSocket functions from a non-WebSocket context. If you use a proxy server, check in the configuration if WebSockets are enabled.

**Examples**

**Basic Example**

The following chapter explains how to create a simple basic web application with WebSockets. You can find another example in the BaseX source code.

First of all, you have to ensure that the `WsServlet` is enabled in your `web.xml` file. It will be enabled if you use the standard configuration of BaseX.

For establishing a connection to the WebSocket server, it is necessary that the server provides at least one function annotated with a WebSocket annotation. Let’s start by using the annotation `%ws:connect('/')`. In the connect function, a bidirectional communication with the client can be initialized: attributes such as the id and name of a client can be set, or a welcome message can be emitted to other connected users, and so on.

```xml
declare
  %ws:connect('/')

function example:connect() as empty-sequence() {
}
```

The connect function is sufficient for creating the persistent client/server connection. In order to something sensible with the connection, you should implement a function annotated with `%ws:message('"/"')`:

```xml
```
import module namespace ws = 'http://basex.org/modules/ws'

declare
    %ws:message('/', '{$message}')
function example:message(
    $message  as xs:string
) as empty-sequence() {
    ws:emit($message)
};

In the function above, the WebSocket Module is imported, and the function ws:emit is used for forwarding the message to all connected clients.

The following client-side code demonstrates a basic application of the WebSocket connection:

```javascript
var ws = new WebSocket("ws://localhost:8080/ws");

ws.onmessage = function(event) {
    alert(event.data);
};

function send(message) {
    ws.send(message);
};
```

The send function can be called to pass on a string to the server.

There are no heart-beats in this example. This means that the connection is terminated if nothing happens for 5 minutes (standard timeout). It will also be closed if you send a message that exceeds the standard text size.

**Chat Application**

In the full distributions of BaseX, you will find a little self-contained chat application that demonstrates how WebSockets can be used in practice.

**Changelog**

WebSockets were introduced with Version 9.1.
Chapter 85. 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.

With BaseX 10, results in the rest namespace will be returned unprefixed:

<!-- before -->
<rest:databases xmlns:rest="http://basex.org/rest"/>
<!-- before -->
<rest:databases xmlns="http://basex.org/rest"/>

URL Architecture

A request to the root URL returns all available databases:

http://localhost:8984/rest

<database name="articles" xmlns="http://basex.org/rest">
  <dir>binnaries</dir>
  <resource type="xml" content-type="application/xml" size="77192">1973-02-08-XLTP325.XML</resource>
  ...
</database>

The resources of a database (directories, resource metadata) are listed if a database and an optional directory path is specified:

http://localhost:8984/rest/articles

<database name="articles" xmlns="http://basex.org/rest">
  <dir>binnaries</dir>
  <resource type="xml" content-type="application/xml" size="77192">1973-02-08-XLTP325.XML</resource>
  ...
</database>

The dir elements were introduced with Version 10. Before, information on all resources was listed that were located in the specified path or any of its subdirectories.

If the path to a single resource is specified, the resource itself will be returned:

http://localhost:8984/rest/articles/1973-02-08-XLTP325.XML

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 set as 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). Similar to query, a database or database path is set as context.

**Examples**

• Lists all resources found in the tmp path of the factbook database:
  ```
  http://localhost:8984/rest/factbook/tmp
  ```

• Returns the number of documents in a database:
  ```
  http://localhost:8984/rest/database?
  query=count(.)
  ```

• 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:
  ```
  ```

• 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 name of the root element determines how the body will be evaluated:

• **commands**: Run Command Script

• **query**: Execute XQuery expression

• **run**: Run server-side file (query or command script)

• **command**: Execute single command

The root element may be bound to the optional REST namespace. Existing command scripts can be sent to the server without any modifications:

• Create an empty database and return database information:

```xml
<commands>
  <create-db name='db'/>
</commands>
```
For the other commands, the following child elements are supported:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Required; contains the query string, command string, or file to be run</td>
</tr>
<tr>
<td>parameter</td>
<td>Serialization parameter (with @name and @value attributes)</td>
</tr>
<tr>
<td>option</td>
<td>Database option (with @name and @value attributes)</td>
</tr>
<tr>
<td>variable</td>
<td>Variable bindings</td>
</tr>
<tr>
<td>context</td>
<td>Initial context item</td>
</tr>
</tbody>
</table>

Examples

- Return the first five city names of the factbook database:

  ```xml
  <rest xmlns="http://basex.org/rest">
    <rest><![CDATA[ (//city/name)[position() <= 5] ]]></text>
  </rest>
  ```

- Return string lengths of all text nodes that are found in the node that has been specified as initial context node:

  ```xml
  <query>
    <text>for $i in ./text() return string-length($i)</text>
    <context>
      <xml>
        <text>Hello</text>
        <text>World</text>
      </xml>
    </context>
  </query>
  ```

- Return the registered database users encoded in ISO-8859-1:

  ```xml
  <command>
    <text>show users</text>
    <parameter name='encoding' value='ISO-8859-1'/>
  </command>
  ```

- Create a new database from the specified input and preserve all whitespaces:

  ```xml
  <command>
    <text>create db test http://files.basex.org/xml/xmark.xml</text>
    <option name='chop' value='false'/>
  </command>
  ```

- Bind value to the $person variable and run query find-person.xq, which must be located in the directory specified by WEBPATH:

  ```xml
  <run>
    <variable name='person' value='Johannes Müller'/>
    <text>find-person.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.
There are two ways to store non-XML data in BaseX:

- **Store as Raw Data**: If application/octet-stream is chosen as content-type, the input is added as Binary Data.
- **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
```

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 $a as xs:integer external;
    declare variable $b 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`, `adaptive`, `baseX` → `application/xml`
   - `xhtml` → `text/html`
   - `html` → `text/html`
   - `text` → `text/plain`
   - `json` → `application/json`
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:

```
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:

```java
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 username 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):

```java
// 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"
REST

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 10.0

• Updated: Results in the rest namespace will be returned unprefixed.

• Updated: dir elements are returned when listing the contents of a database.

Version 9.0

• Added: Support for command scripts in the POST Method.

• Updated: The REST namespace in the POST Method has become optional.

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 adress 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 86. 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.

The implementation in BaseX is based on the Milton library. Currently, only Basic Authentication is supported.

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).

Authentication

The WebDAV service uses the database user credentials to perform authentication and authorization. A default user can be defined in the configuration as described here.

Root Directory

In the WebDAV root directory, all existing databases are listed. As new resources can only be stored inside a 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.

Resources

XML Documents

Uploaded files that start with an angle bracket will be stored as XML files. XML entities will be decoded during this process.

If a file is downloaded, the characters with the following code points will be encoded as entities:

- 160 (non-breaking space)
- 8192–8207, 8232–8239, 8287–8303 (see General Punctuation)

Binary Files

If XML parsing files, or if the first character of the input is no angle bracket, the file will be stored as binary 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 87. 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."

[Image]

• Click "Next", select "Choose a custom network location" and click "Next" again.

[Image]

• Enter the URL address of the BaseX WebDAV Server (e.g. http://localhost:8984/webdav) and click "Next".
If a message saying that the folder is not valid, this is because Microsoft WebClient is not configured to use Basic HTTP authentication. Please check out the following StackOverflow entry 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 88. WebDAV: Windows XP

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows XP.

• In the "My Network Places" view, double click on "Add Network Place":

• Confirm the upcoming introductory dialog:

• Select "Choose another network location" in the next dialog:
WebDAV: Windows XP

Where do you want to create this network place?

Select a service provider. If you do not have a membership with the provider you select, the wizard will help you create an account. To just create a shortcut, click "Choose another network location."

Service providers:

- MSN Communities
  - Share your files with others, or store them for your personal use.

- Choose another network location
  - Specify the address of a Web site, network location, or FTP site.

• Next, specify the BaseX WebDAV URL:

What is the address of this network place?

Type the address of the Web site, FTP site, or network location that this shortcut will open.

Internet or network address:
http://localhost:8984/webdav/

Browse...

View some examples.

• Enter the user/password combination to connect to the WebDAV service:
WebDAV: Windows XP

• Assign a name to your WebDAV connection:

• Finish the wizard:

• You can now see all BaseX databases in the Windows Explorer:
## WebDAV: Windows XP

![WebDAV Interface](image)

<table>
<thead>
<tr>
<th>Name</th>
<th>Internet Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>factbook</td>
<td><a href="http://localhost:8994/webdav/factbook">http://localhost:8994/webdav/factbook</a></td>
<td>Web Folder</td>
</tr>
</tbody>
</table>

1 objects
Chapter 89. 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: Mac OSX

```xml
<country id="FR_136" country="France" population="6039400" location="Paris" timezone="CET" currency="EUR" government="emerging democracy" car_code="FR" />
<city id="FR_136" country="France" population="1200000" location="Lyon" timezone="CET" currency="EUR" government="emerging democracy" car_code="FR" />
<city id="FR_30514" country="France" population="200000" location="Toulouse" timezone="CET" currency="EUR" government="emerging democracy" car_code="FR" />
```

Image showing a file explorer window with a file named `facebook.xml` and an open XML editor window.
Chapter 90. 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:
WebDAV: GNOME

- After clicking "Connect" the databases can be browsed:

![WebDAV GNOME screenshot showing database browser]

```
records.xml
```

1 item
Chapter 91. 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/webdav):

• Enter the user credentials:

• After clicking "OK" the databases can be browsed:
WebDAV: KDE
Part X. Client APIs
Chapter 92. Clients

Read this entry online in the BaseX Wiki.

The conventions for the client functions were revised:

• The `replace` and `store` functions have been renamed to `put` and `putBinary`.
• Existing client implementations are working without changes unless they haven’t been upgraded.

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

Please see the Server Protocol for more information on the available commands. Currently, we offer bindings for the following programming languages:

**BaseX 7.x, BaseX 8.x and later**

- **Java**: The default implementation
- **C++**: contributed by Karim Salama (based on SDL2_net)
- **C++**: contributed by Jean-Marc Mercier (based on libboost)
- **C#**: contributed by the BaseX Team and Martín Ferrari
- **C**: contributed by the BaseX Team
- **Golang**: contributed by Christian Baune
- **Erlang**: contributed by Zachary Dean
- **node.js**: contributed by Andy Bunce
- **Perl**: contributed by the BaseX Team
- **PHP**: updated by James Ball
- **Python**: contributed by Hiroaki Itoh
- **Python**, using BaseX REST services: contributed by Luca Lianas
- **R**: contributed by Ben Engbers
- **Raku**: contributed by Wayland
- **Ruby**: contributed by the BaseX Team

**BaseX 7.x (outdated)**

- **ActionScript**: contributed by Manfred Knobloch
- **Haskell**: contributed by Leo Wörteler
- **Lisp**: contributed by Andy Chambers
- **node.js**: contributed by Hans Hübner (deviating from client API)
- **Qt**: contributed by Hendrik Strobelt
- **Rebol**: contributed by Sabu Francis
- **Scala**: contributed by Manuel Bernhardt
- **Scala** (simple implementation)
- **VB**: 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!

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 10.0

- Updated: The replace and store functions have been renamed to put and putBinary.

Version 8.0

- Updated: cram-md5 replaced with digest authentication.
Chapter 93. 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 function returns the whole result. With the `info()` function, you can request some information on your executed process. If an error occurs, an exception with the error message will be thrown.

Usage

The standard execution works as follows:

1. Create a new session instance with hostname, port, username and password.
2. Call the `execute()` function of the session with the database commands as argument.
3. Receive the result of a successfully executed command. If an error occurs, an exception is thrown.
4. Optionally, call `info()` to get some process information
5. Continue using the client (back to 2.), or close the session.

Example in PHP

Taken from our repository:

```php
<?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", "..." );

    // 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 94. Query Mode

The query mode of the Clients allows you to bind external variables to a query and evaluate the query in an iterative manner. The query() function of the Session instance returns a new query instance.

Usage

The query execution works as follows:

1. Create a new session instance with hostname, port, username and password.
2. Call query() with your XQuery expression to get a query object.
3. Optionally bind variables to the query with one of the bind() functions.
4. 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
<?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", "...");

    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();
    }
}
```

445
} 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 95. 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, username 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()`, `put()` and `putbinary()` methods pass on input streams to the corresponding database commands. The input can be a UTF-8 encoded XML document, a binary resource, or any other data (such as JSON or CSV) that can be successfully converted to a resource by the server.

• 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

• \( \text{xx} \) : single byte.

• \( \{\ldots\} \) : utf8 strings or raw data, suffixed with a \( \text{\00} \) byte. To avoid confusion with this end-of-string byte, all transferred \( \text{\00} \) and \( \text{\FF} \) bytes are prefixed by an additional \( \text{\FF} \) byte.

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: \( \text{realm:nonce} \)

3. Client sends the username and a hash value. The hash is composed of the md5 hash of

   a. the md5 hash of the username, realm, and password (all separated by a colon), and
b. the nonce: \{username\} \{md5(md5(username:realm:password) + nonce)\}

4. Server replies with \00 (success) or \01 (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 username 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 \00 (success) or \01 (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):

<table>
<thead>
<tr>
<th>Command</th>
<th>Client Request</th>
<th>Server Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMAND</td>
<td>{command}</td>
<td>{result} {info} \00</td>
<td>Executes a database command.</td>
</tr>
<tr>
<td>QUERY</td>
<td>\00 {query}</td>
<td>{id} \00</td>
<td>Creates a new query instance and returns its id.</td>
</tr>
<tr>
<td>CREATE</td>
<td>\08 {name} {input} {info} \00</td>
<td></td>
<td>Creates a new database with the specified input (may be empty).</td>
</tr>
<tr>
<td>ADD</td>
<td>\09 {name} {path} {input} {info} \00</td>
<td></td>
<td>Adds a new document to the opened database.</td>
</tr>
<tr>
<td>PUT</td>
<td>\0C {path} {input} {info} \00</td>
<td></td>
<td>Puts (adds or replaces) an XML document resource in the opened database.</td>
</tr>
<tr>
<td>PUTBINARY</td>
<td>\0D {path} {input} {info} \00</td>
<td></td>
<td>Puts (adds or replaces) a binary resource in the opened database.</td>
</tr>
<tr>
<td># error</td>
<td></td>
<td>{partial result} {error} \01</td>
<td>Error feedback.</td>
</tr>
</tbody>
</table>

**Query Command Protocol**

Queries are referenced via an id, which has been returned by the QUERY command (see above).

<table>
<thead>
<tr>
<th>Query Command</th>
<th>Client Request</th>
<th>Server Response</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSE</td>
<td>\02 {id}</td>
<td>\00 \00</td>
<td>Closes and unregisters the query with the specified id.</td>
</tr>
</tbody>
</table>
As can be seen in the table, all results end with a single \00 byte, which indicates that the process was successful. If an error occurs, an additional byte \01 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 \01 bytes.

- Item types are specified by appending \02 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, \01, 789 and \00 (xs:integer may be specified via the (type) argument).

- the two items xs:integer(123) and xs:string('ABC') are encoded as 123, \02, xs:integer, \01, ABC, \02, xs:string and \00.
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 username "jack": 6A 61 63 6B 00 #
- **Client** sends hash: \(\text{md5(md5("jack:BaseX:topsecret") + "1369578179679") = ca664a31f8deda9b71ea3e79347f66666": 63 61 36 ... 00 #}\)
- **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 ...
- **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 \00 byte instead of a new result, which indicates that no more results can be expected: # 00
- **Server** sends the error code \01 and the error message ("Stopped at..."): # 01 53 74 6f ...
- **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, username and password:
  \[\text{Session(String host, int port, String name, String password)}\]
- Execute a command and return the result:
  \[\text{String execute(String command)}\]
- Return a query instance for the specified query:
  \[\text{Query query(String query)}\]
- Create a database from an input stream:
  \[\text{void create(String name, InputStream input)}\]
- Add a document to the current database from an input stream:
  \[\text{void add(String path, InputStream input)}\]
Server Protocol

- Put a document with the specified input stream: `void put(String path, InputStream input)`
- Put a binary resource at the specified path: `void putBinary(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 96. Server Protocol: Types

This article lists extended type information that is returned by the Server Protocol.

**XDM Metadata**

In most cases, the XDM metadata is nothing else than the Type ID. There are three exceptions: 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`:

<table>
<thead>
<tr>
<th>Type ID</th>
<th>Node Kind/Item Type</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Function item</td>
<td><code>function</code></td>
</tr>
<tr>
<td>8</td>
<td>node()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>9</td>
<td>text()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>10</td>
<td>processing-instruction()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>11</td>
<td>element()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>12</td>
<td>document-node()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>13</td>
<td>document-node(element())</td>
<td><code>node</code></td>
</tr>
<tr>
<td>14</td>
<td>attribute()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>15</td>
<td>comment()</td>
<td><code>node</code></td>
</tr>
<tr>
<td>32</td>
<td>item()</td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>33</td>
<td><code>xs:untyped</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>34</td>
<td><code>xs:anyType</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>35</td>
<td><code>xs:anySimpleType</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>36</td>
<td><code>xs:anyAtomicType</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>37</td>
<td><code>xs:untypedAtomic</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>38</td>
<td><code>xs:string</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>39</td>
<td><code>xs:normalizedString</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>40</td>
<td><code>xs:token</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>41</td>
<td><code>xs:language</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>42</td>
<td><code>xs:NM_TOKEN</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>43</td>
<td><code>xs:Name</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>44</td>
<td><code>xs:NCName</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>45</td>
<td><code>xs:ID</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>46</td>
<td><code>xs:IDREF</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>47</td>
<td><code>xs:ENTITY</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>48</td>
<td><code>xs:float</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>49</td>
<td><code>xs:double</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>50</td>
<td><code>xs:decimal</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td>51</td>
<td><code>xs:precisionDecimal</code></td>
<td><code>atomic value</code></td>
</tr>
<tr>
<td></td>
<td>xs:integer</td>
<td>atomic value</td>
</tr>
<tr>
<td>---</td>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>52</td>
<td>xs:nonPositiveInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>53</td>
<td>xs:negativeInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>54</td>
<td>xs:long</td>
<td>atomic value</td>
</tr>
<tr>
<td>55</td>
<td>xs:int</td>
<td>atomic value</td>
</tr>
<tr>
<td>56</td>
<td>xs:short</td>
<td>atomic value</td>
</tr>
<tr>
<td>57</td>
<td>xs:byte</td>
<td>atomic value</td>
</tr>
<tr>
<td>58</td>
<td>xs:nonNegativeInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>59</td>
<td>xs:unsignedLong</td>
<td>atomic value</td>
</tr>
<tr>
<td>60</td>
<td>xs:unsignedInt</td>
<td>atomic value</td>
</tr>
<tr>
<td>61</td>
<td>xs:unsignedShort</td>
<td>atomic value</td>
</tr>
<tr>
<td>62</td>
<td>xs:unsignedByte</td>
<td>atomic value</td>
</tr>
<tr>
<td>63</td>
<td>xs:positiveInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>64</td>
<td>xs:duration</td>
<td>atomic value</td>
</tr>
<tr>
<td>65</td>
<td>xs:yearMonthDuration</td>
<td>atomic value</td>
</tr>
<tr>
<td>66</td>
<td>xs:dateTime</td>
<td>atomic value</td>
</tr>
<tr>
<td>67</td>
<td>xs:dateTimeStamp</td>
<td>atomic value</td>
</tr>
<tr>
<td>68</td>
<td>xs:date</td>
<td>atomic value</td>
</tr>
<tr>
<td>69</td>
<td>xs:time</td>
<td>atomic value</td>
</tr>
<tr>
<td>70</td>
<td>xs:gYearMonth</td>
<td>atomic value</td>
</tr>
<tr>
<td>71</td>
<td>xs:gYear</td>
<td>atomic value</td>
</tr>
<tr>
<td>72</td>
<td>xs:gMonthDay</td>
<td>atomic value</td>
</tr>
<tr>
<td>73</td>
<td>xs:gDay</td>
<td>atomic value</td>
</tr>
<tr>
<td>74</td>
<td>xs:gMonth</td>
<td>atomic value</td>
</tr>
<tr>
<td>75</td>
<td>xs:boolean</td>
<td>atomic value</td>
</tr>
<tr>
<td>76</td>
<td>base64Binary</td>
<td>atomic value</td>
</tr>
<tr>
<td>77</td>
<td>xs:base64Binary</td>
<td>atomic value</td>
</tr>
<tr>
<td>78</td>
<td>xs:hexBinary</td>
<td>atomic value</td>
</tr>
<tr>
<td>79</td>
<td>xs:anyURI</td>
<td>atomic value</td>
</tr>
<tr>
<td>80</td>
<td>xs:QName</td>
<td>atomic value</td>
</tr>
<tr>
<td>81</td>
<td>xs:NOTATION</td>
<td>atomic value</td>
</tr>
</tbody>
</table>
Chapter 97. 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 ()

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.

**XQJ API (closed source)**

The implementation of the BaseX XQJ API has been written by Charles Foster. The basex-examples repository contains various examples on how to use XQJ.
Part XI. Extensions
BaseX server can be configured to run as an always-on service in Windows (or daemon in Linux) using **YAJSW**.

### Some basics of YAJSW

- Each service running with YAJSW has a configuration file which lives in the `conf` folder.
- Installing and controlling services is done from the command line. Run a command prompt as administrator, then navigate to the folder where you placed YAJSW, e.g. `cd C:\Programs\yajsw\yajsw-beta-12.05`
- If you need to change configuration of a service follow this sequence:
  1. Stop the service `java -jar wrapper.jar --stop conf\wrapper.name.conf`
  2. Remove the service `java -jar wrapper.jar --remove conf\wrapper.name.conf`
  3. Make your changes to the wrapper or application configuration.
  4. Install the service `java -jar wrapper.jar --install conf\wrapper.name.conf`
  5. Start the service `java -jar wrapper.jar --start conf\wrapper.name.conf`

YAJSW comes with some helpful convenience scripts in the 'bat' and 'bin' folders. This set of instructions does not use these convenience scripts.

### Gather the files

- Download the latest version of [BaseX](https://www.basex.com/). Select the Zip Archive.
- Download the latest version of [YAJSW](https://www.yajsw.com/)
- Download the latest version of [Java](https://www.oracle.com/java/index.html).

### Install BaseX as a Windows Service

The instructions on this page are known to work using Windows Server 2012R2, BaseX 8.4.2, YAJSW 12.05 beta, Java 1.8.0_77 64-bit from Oracle.

### Install Java

Install Java using the Java installer for your operating system. Use a 64-bit version if you can.

### Put files into position

These instructions assume you will be placing BaseX and YAJSW in C:\Programs, but you can choose a different location.

1. Create folder `C:\Programs`
2. Extract YAJSW to `C:\Programs\yajsw\yajsw-beta-12.05`
3. Extract BaseX to `C:\Programs\BaseX\basex`

See [Database Server](https://basex.com) for information on how to pick an initial admin password.
Install BaseX as a service

Create wrapper config file wrapper.basex.conf and place it in YAJSW's conf folder. You can use the example below. You may need to modify this example to:

- Specify the location of java.exe
- Change the amount of memory available to BaseX from 1024m (for example, 512m or 2048m)

```bash
# YAJSW configuration for BaseX
wrapper.java.command=C:/ProgramData/Oracle/Java/javapath/java.exe
wrapper.working.dir=C:\Programs\BaseX\basex
wrapper.java.app.mainclass=org.basex.BaseXHTTP
wrapper.java.classpath.1 = .\BaseX.jar
wrapper.java.classpath.2 = .\lib\*.jar
wrapper.java.classpath.3 = .\lib\custom\*.jar
wrapper.java.additional.1 = -Xmx1024m
wrapper.java.additional.2 = -Dfile.encoding=utf-8
wrapper.ntservice.name=BaseX
wrapper.ntservice.displayname=BaseX
wrapper.ntservice.description=BaseX XQuery database
wrapper.ntservice.starttype=DELAYED_AUTO_START
wrapper.console.loglevel=INFO
wrapperlogfile=${wrapper.working.dir}\data\wrapper-basex.log
wrapper.logfile.maxsize=10m
wrapper.logfile.maxfiles=10
wrapper.on_exit.0=SHUTDOWN
wrapper.on_exit.default=RESTART
```

After you have created the wrapper configuration file:

1. Open a command prompt as administrator
2. Navigate to the YAJSW folder: cd C:\Programs\yajsw\yajsw-beta-12.05
3. Install the service: java -jar wrapper.jar --install conf\wrapper.basex.conf
4. Start the service: java -jar wrapper.jar --start conf\wrapper.basex.conf

BaseX server is now running as a service, and will start automatically when Windows starts.
Chapter 99. Android

Read this entry online in the BaseX Wiki.

It is possible to create an Android port of BaseX. The present tutorial outlines the creation of a BaseX Android library, which can be used in any other application project.

For the creation of the library the IDE Android Studio is used, but the steps are more or less equal using the Eclipse IDE.

Creating the Android Library Project

The first step is to create an Android library project, which will be later modified to represent the BaseX Android library.

In Android Studio the 'Create New Project' menu item needs to be chosen. In order to this the displayed window appears.

It is important that the minimum Android version is Gingerbread 2.3.3, because of some String methods used in BaseX which are not supported by Android versions older than Gingerbread.

To create an Android library project, the 'Mark this project as library' item need to be checked. An Android library is not executable and therefore does not need the creation of an Activity, which is the reason why this item is unchecked in the picture above.

After finishing the dialog Android Studio creates an empty library project with all needed folders and files.

The next step is to copy the BaseX code into the created project folder 'src/main/java'.

Except the package 'gui' and the Java file 'BaseXGui.java' inside the 'src.main.java.org.basex'[1] package can be copied into the project folder. Android does not support Java AWT and Swing, which is the reason for not copying the gui package.
Adjusting the Code

After successfully copying the corresponding BaseX packages and java files into the created Android library project a few adjustments have to be done in order to get a working Android library.

At this moment the BaseX source code is presented in the Android library project as well as an empty android package, as it is shown in the following image.
Android
In the empty android package a new Java class needs to be created, this class is used to create the necessary BaseX files and communicate with BaseX. This class needs the data directory of the application for storing the corresponding BaseX files. This files should be stored in the apps /data/data/.. folder which is only accessible from the application. This information is only available inside the applications context and not inside a library project, therefore it is necessary to pass this information to this class at the constructor call. The following source code shows a minimal example for a BaseX class.

```java
public class BaseXDatabase {
    private Context basexContext = null;

    public BaseXDatabase(String data_dir) {
        basexContext = new Context(data_dir);
    }
}
```

This class can be called in every Android application which uses the BaseX library with the following call, for example:

```java
BaseXDatabase baseXDatabase = new BaseXDatabase(getApplicationInfo().dataDir);
```

At the moment it is not possible to use the BaseX library, therefore more adjustments have to be done in the BaseX code.

First it is necessary to add an additional constructor to the Context class to create the BaseX files in the right directory and adjust the default constructor of it. The following code shows the changes inside the Context.java file:

```java
public Context(String data_dir) {
    this(true, (Prop.HOME = data_dir + "/"), (Prop.USERHOME = data_dir + "/"));
    File dir = new File(Prop.HOME, "basex/data");
    if(!dir.exists()) {
        if(!dir.mkdir()) {
            android.util.Log.i("BASEX", "CREATING BASEX DIRECTORIES");
        }
    }
}
```

```java
private Context(final boolean file, String home, String userhome) {
    this(new MainProp(file));
}
```

As shown in the adjustment above, it is necessary to set the two variables 'Prop.HOME' and Prop.USERHOME' during the constructor call. In the BaseX code those variables are final, which need also be changed in order to set them during the call.

The reason for this change is that the in BaseX used System.getProperty(user.dir) returns an empty string in Android.

The next adjustment, which needs to be done, is to remove not supported packages inside the BaseX code. Therefore the package 'org.basex.query.util.crypto' need to be removed, because it uses external packages which are not supported by Android. The class which uses these files can be found inside the FNCrypto.java file in the 'query.func' package. This file needs to be deleted as well as its usage inside the Function.java file, which can also be found inside the 'query.func' package. The following lines need to be removed:

```java
/** XQuery function. */
CRYPTO_HMAC(FNCrypto.class, "hmac(message,key,algorithm[,encoding])",(arg(STR, STR, STR, STR_ZO), STR)),
/** XQuery function. */
CRYPTO_ENCRYPT(FNCrypto.class, "encrypt(input,encryption,key,algorithm)",(arg(STR, STR, STR, STR), STR)),
/** XQuery function. */
CRYPTO_DECRYPT(FNCrypto.class, "decrypt(input,type,key,algorithm)",(arg(STR, STR, STR, STR), STR)),
```

462
/* XQuery function. */
CRYPTO_GENERATE_SIGNATURE(FNCrypto.class, "generate-signature" +
  "(input,canonicalization,digest,signature,prefix,type[,item1][,item2])",
  arg(NOD, STR, STR, STR, STR, ITEM_ZO, ITEM_ZO), NOD),
/* XQuery function. */
CRYPTO_VALIDATE_SIGNATURE(FNCrypto.class, "validate-signature(node)", arg(NOD),
BLN),
URIS.put(FNCrypto.class, CRYPTOURI);

The result of this adjustment is, that it is now possible to use BaseX as an Android library, with the lack of support
of the following XQuery functions:

- hmac(string,string,string[,string])
- encrypt(string,string,string,string)
- decrypt(string,string,string,string)
- generate-signature(node,string,string,string,string,[string][,item][,item])
- validate-signature(node)

Using the BaseX Android Library

To use the BaseX library the above created BaseXDatabase class can be extended with additional methods which
are delegating requests to the BaseX database and return the results.

An example of this can be seen in the following code:

```java
public String executeXQuery(String query) throws IOException {
    if(basexContext != null)
        return new XQuery(query).execute(basexContext);
    else
        Log.e("BaseXDatabase", "No context");
        return "";
}
```

This methods of the BaseXDatabase class can now be used in every Android application which includes the created
BaseX Android library.

It is possible to create a .jar, or an .aar file out of the BaseX library, by just building the source code. This file need
to be copied inside the lib folder of the Android project which wants to use the library. Additionally the build file
of the application needs to be adjusted to use the library.

Using Gradle, the Android build system, it can be done by adding the following line to the gradle build file. This
tells the build system that every library, inside the libs folder, is being compiled into the projects file.

```gradle
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
}
```
Part XII. Advanced User's Guide
Chapter 100. 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 & Parsing

- **Catalog Resolver** Information on entity resolving
- **Storage Layout** : How data is stored in the database files
- **Binary Data** : How to store and use binary data
- **Parsers** : How different input formats can be converted to XML

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
Chapter 101. 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 mapping system IDs (DTD locations)
and URIs to local resources when parsing and transforming XML data.
Changed with Version 10:
• The Java 11: XML Catalog API is used to resolve references to external resources.
• As an alternative, Norman Walsh’s Enhanced XML Resolver is utilized if it is found in the classpath.
• The Apache-maintained XML Commons Resolver has become obsolete.
• If enabled, a catalog is universally applied for resolving:
• entities (when parsing XML documents);
• URIs (for documents, module imports, XSL transformations);
• resources (when validating documents).

Introduction
XML documents often rely on Document Type Definitions (DTDs). 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 from the W3C’s server 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.
To address these issues, the XML Catalogs Standard defines an entity catalog that maps both external identifiers
and arbitrary URI references to URI references.
Another application for XML catalogs is to provide local resources for reusable XSLT stylesheet libraries that are
imported from a canonical location. This is described in greater detail in the following section.

Usage
System ID (DTD Location) Rewrites
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:
<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/. For example, if the following XML file is parsed:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

466


Catalog Resolver

The XHTML DTD xhtml1-transitional.dtd and all its linked resources will now be loaded from the specified path.

The catalog file etc/w3-catalog.xml in the full distributions can be used out of the box. It defines rewritings for some common W3 DTD files.

**URI Rewrites**

Consider a library of reusable XSLT stylesheets. For performance reasons, this library will be cached locally. However, the import URI for a given stylesheet should always be the same, independent of the accidental relative or absolute path that it is stored at locally. Example:

```xml
<xsl:import href="http://acme.com/xsltlib/acme2html/1.0/acme2html.xsl"/>
```

The XSLT stylesheet might not even be available from this location. The URI serves as a canonical location identifier for this XSLT stylesheet. A local copy of the acme2html/1.0/ directory is expected to reside somewhere, and the location of this directory relative to the local XML catalog file is specified in an entry in this catalog, like this:

```xml
<rewriteURI uriStartString="http://acme.com/xsltlib/acme2html/1.0/"
               rewritePrefix="../acmehtml10/"/>
```

This way, XSLT import URIs don’t have to be adjusted for the relative or absolute locations of the XSLT library’s local copy.

The same URI rewriting works for resources retrieved by the `doc()` function from within an XSLT stylesheet. See XSLT Module for details on how to invoke XSLT stylesheets from within BaseX.

NOTE: This URI rewriting is currently restricted to XSLT stylesheets. It has neither been enabled yet for the XQuery function `doc()` nor for XSD schema locations.

**GUI Mode**

When running BaseX in GUI mode, enable DTD parsing and 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, enable the `DTD` option and assign the path to your XML catalog file to the `CATALOG` option. All subsequent commands for adding documents will use the specified catalog file to resolve entities.

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, they are resolved against the working directory of the *server*.

**Additional Notes**

Entity resolving only works if the *internal XML parser* is switched off (which is the default case).

By default, an error is raised if the catalog resolution fails. The runtime properties of the catalog resolver can be changed by setting system properties, either on startup...

```java
java -Djavax.xml.catalog.resolve=continue ... org.basex.BaseX
```

...or via XQuery:

```xquery
Q(java:System)setProperty('javax.xml.catalog.resolve', 'continue'), ...
```

See Java 11: XML Catalog API for more information.
When using a catalog within an XQuery Module, the global \texttt{db:catalog} option may not be set in this module. You can set it via pragma instead:

\begin{verbatim}
(# db:catalog xmlcatalog/catalog.xml #) {
    xslt:transform(db:get('acme_content')[1], '../acmecustom/acmehtml.xsl')
}
\end{verbatim}

It is assumed that this stylesheet \texttt{../acmecustom/acmehtml.xsl} (location relative to the current XQuery script or module) imports \texttt{acme2html/1.0/acme2html.xsl} by its canonical URI that will be resolved to a local URI by the catalog resolver.

Please note that since catalog-based URI rewriting does not work yet within URIs accessed from XQuery, you cannot give a canonical location that needs to be catalog-resolved as the second argument of \texttt{xslt:transform()}. The catalog location in the pragma can be given relative to the current working directory (the directory returned by \texttt{file:current-dir()}) or as an absolute operating system path. The catalog location in the pragma is not an XQuery expression; no concatenation or other operations may occur in the pragma, and the location string must not be surrounded by quotes.

\section*{Changelog}

Version 10.0

- Updated: \texttt{CATFILE} option renamed to \texttt{CATALOG}. 

Chapter 102. Storage Layout

This article is part of the Advanced User's Guide. It presents some low-level details on how data is stored in the database files.

Data Types

The following data types are used for specifying the storage layout:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Example (native → hex integers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num</td>
<td>Compressed integer (1-5 bytes), specified in Num.java</td>
<td>15 → 0F; 511 → 41 FF,</td>
</tr>
<tr>
<td>Token</td>
<td>Length (Num) and bytes of UTF8 byte representation</td>
<td>Hello → 05 48 65 6c 6c 6f</td>
</tr>
<tr>
<td>Double</td>
<td>Number, stored as token</td>
<td>123 → 03 31 32 33</td>
</tr>
<tr>
<td>Boolean</td>
<td>Boolean (1 byte, 00 or 01)</td>
<td>true → 01</td>
</tr>
<tr>
<td>Nums, Tokens, Doubles</td>
<td>Arrays of values, introduced with the number of entries</td>
<td>1,2 → 02 01 31 01 32</td>
</tr>
<tr>
<td>TokenSet</td>
<td>Key array (Tokens), next/bucket/size arrays (3x Nums)</td>
<td></td>
</tr>
</tbody>
</table>

Database Files

The following tables illustrate the layout of the BaseX database files. All files are suffixed with .basex.

Metadata, Name/Path/Doc Indexes: inf

<table>
<thead>
<tr>
<th>Description</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Metadata</td>
<td>1. Key/value pairs, in no particular order (Token/Token):</td>
</tr>
<tr>
<td></td>
<td>• Examples: FNAME, TIME, SIZE, ...</td>
</tr>
<tr>
<td></td>
<td>• PERM → Number of users (Num), and name/password/permission values for each user (Token/Token/Num). Empty key as finalizer</td>
</tr>
<tr>
<td>2. Main memory indexes</td>
<td>1. Key/value pairs, in no particular order (Token/Token):</td>
</tr>
<tr>
<td></td>
<td>• TAGS → Element Name Index • ATTS → Attribute Name Index • PATH → Path Index • NS → Namespaces • DOCS → Document Index2. Empty key as finalizer</td>
</tr>
<tr>
<td>2 a) Name Index</td>
<td>1. Token set, storing all names (TokenSet).</td>
</tr>
<tr>
<td></td>
<td>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. Maximum text length (Double; legacy, could be Num)</td>
</tr>
<tr>
<td>2 b) Path Index</td>
<td>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)</td>
</tr>
</tbody>
</table>
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)

2 d) Document Index

Array of integers, representing the distances between all document pre values (Nums)

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, the Token Index in tokl and tokr. All have the same layout.

For a more detailed discussion and examples of these file formats please see Index File Structure.

Document Path Index: pth

Provides an index of all the document paths in the database. For databases with a large number of paths this file can be quite large so it is only generated the first time a function requesting a path lookup is run. For databases where path lookups are never used this file will not exist.

Note: On Windows/Mac systems this file is case insensitive (all paths are lower case). On UNIX-like systems this file is case sensitive. The behaviour of path lookups will vary between systems. Copying this file between system types may lead to unexpected behaviour.

ID/Pre Mapping: idp

This file is only created if incremental indexing (UPDINDEX) is enabled for a database. It is used to provide a quick look up of the pre value for a database node id.

Full-Text Fuzzy Index: ftxx, ftxy, ftxz

...may soon be reimplemented.
Chapter 103. 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"
```

<table>
<thead>
<tr>
<th>PRE</th>
<th>DIS</th>
<th>SIZ</th>
<th>ATS</th>
<th>ID</th>
<th>NS</th>
<th>KIND</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DOC</td>
<td>db.xml</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>ELEM</td>
<td>xml</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>TEXT</td>
<td>HiThere</td>
</tr>
</tbody>
</table>

**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"
```

<table>
<thead>
<tr>
<th>PRE</th>
<th>DIS</th>
<th>SIZ</th>
<th>ATS</th>
<th>ID</th>
<th>NS</th>
<th>KIND</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>DOC</td>
<td>db.xml</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>ELEM</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>ELEM</td>
<td>xml</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>TEXT</td>
<td>HiThere</td>
</tr>
</tbody>
</table>

The `db:node-pre` and `db:node-id` functions can be called to retrieve the `pre` and `id` values of a node, and `db:get-pre` and `db:get-id` can be used to go back and retrieve the original node. By default, `id` lookups are expensive. If UPDINDEX 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:

<table>
<thead>
<tr>
<th>fpre's</th>
<th>addr's</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 13, 257</td>
<td>0, 8192, 4096</td>
</tr>
</tbody>
</table>

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 104. Binary Data

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide.

The BaseX store also provides support for binary resources. A database may contain both XML documents and binary files, which are 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 binary files are stored unchanged in a dedicated subdirectory (called `raw`). Several reasons exist for using the traditional file system as storage:

- **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.

- **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 subdirectories.

- **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 105. Parsers

This article is part of the Advanced User's Guide. 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 alternatives for parsing XML:

- By default, Java's SAXParser is used to parse XML documents.
- The internal, built-in XML parser is more fault-tolerant than Java’s XML parser. It supports standard HTML entities out of the box, and it is faster than the default parser, in particular if small documents are to be parsed. However, the internal parser does not support the full range of DTD features and cannot resolve catalogs.

**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:

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

**XQuery**

The `db:add` and `db:put` 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`:

```xquery
for $file in file:list("2Bimported")
return db:add('DB', $file, '', map { 'intparse': true() })
```

**HTML Parser**

If TagSoup is found in the classpath, 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.).

If TagSoup is not available on a system, the default XML parser will be used. (Only) if the input is well-formed XML, the import will succeed.

**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 project is to follow these steps:

1. Visit MVN TagSoup Repository
2. Click on the version you want

3. On the first tab, you can see an XML snippet like this:

```xml
<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` file into the `<dependencies>` element.

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:

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

**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 this option.
- **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:

```bash
SET PARSER html
SET HTMLPARSER
method=xml,nons=true,nocdata=true,nodefaults=true,nobogons=true,nocolons=true,ignoreable=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 as function arguments:
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.

---

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 an XQuery function. 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:

```
for $file in file:list("2Bimported", false(), "*.csv")
return db:add("DB", $file, ",", map {
    'parser': 'csv',
    'csvparser': map { 'header': true() }
})
```

**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 also be specified via XQuery:

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

**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 106. 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 GUI and the Command-Line Client 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.

Please take care of usual 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.

![Permission Hierarchy Diagram]

User names must follow the valid names constraints, and the database patterns must follow the Glob Syntax.

Operations

For all operations, admin permissions are required:

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

**XQuery**

The available user functions are listed in the User Module:

Create user 'test' with no permissions:

user:create('test', 'top-secret')

Show detailed information about user 'test':

user:list-details()[@name = 'test']

Drop user 'test':

user:drop('test')

**Storage**

The permission file users.xml is stored in the database directory. This file can be manually edited; it will be parsed once when BaseX is started.

Salted SHA256 hashes are 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 AUTHMETHOD is set to Digest.

**Changelog**

Revised in Version 8.0.
Chapter 107. 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.

Introduction

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.

Please note that:

- Locks cannot be synchronized across BaseX instances that run in different JVMs. If concurrent write operations are to be performed, we generally recommend working with the client/server or the HTTP architecture.

- 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 shutdown time. It is advisable to use the CREATE BACKUP command to regularly back up your database. If the worst case occurs, you can try the INSPECT command to check if your database has obvious inconsistencies, and use RESTORE to restore the last backed up version of the database.

XQuery Update

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 support for multiple read and single write operations (using preclaiming and starvation-free two phase locking). This means that:

- Read transactions are executed in parallel.

- If an updating transaction comes in, it will be queued and executed after all previous read transaction have been executed.

- Subsequent operations (read or write) will be queued until the updating transaction has completed.

- Jobs without database access will never be locked. Globally locking jobs can now be executed in parallel with non-locking jobs.

- Each database has its own queue: An update on database A will not block operations on database B. This is under the premise that it can be statically determined, i.e., before the transaction is evaluated, which databases will be accessed by a transaction (see below).

- The number of maximum parallel transactions can be adjusted with the PARALLEL option.

- By default, read transactions are favored, and transactions that access no databases can be evaluated even if the transactions limit has been reached. This behavior can be changed via the FAIRLOCK option.
Limitations

Commands

All commands come with a detector for local locks. Global locking is applied if the glob syntax is used:

- DROP DB new*: Drop all databases starting with the prefix string new.

XQuery

With Version 10, the lock detection has been fundamentally improved, by splitting compilation into multiple steps.

Local locks can be applied if it is possible after compile time to associate all database operations with static databases names:

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>//item</td>
<td>Read lock of the currently opened database</td>
</tr>
<tr>
<td>doc('factbook')</td>
<td>Read lock of the factbook database</td>
</tr>
<tr>
<td>collection('documents/path/to/docs')</td>
<td>Read lock of the documents database</td>
</tr>
<tr>
<td>delete nodes db:get('test')/*/[@type = 'misc']</td>
<td>Write lock of the test database</td>
</tr>
<tr>
<td>declare variables $db external;db:get($db)</td>
<td>Read lock of the database externally bound to $db.</td>
</tr>
<tr>
<td>for $db in ('db1', 'db2')return db:get($db)</td>
<td>Read lock of db1 and db2, as the query is unrolled at compile time.</td>
</tr>
<tr>
<td>let $db := 'test'return insert nodes &lt;test/&gt; into db:get($db)</td>
<td>Read lock of test, as the variable is inlined at compile time.</td>
</tr>
<tr>
<td>sum(1 to 100)</td>
<td>No lock required</td>
</tr>
<tr>
<td>declare variable $SIMULATE := true();if($SIMULATE) then &lt;doc/&gt; else db:get('doc')</td>
<td>No lock required, as the query is simplified to &lt;doc/&gt; at compile time.</td>
</tr>
</tbody>
</table>

A global lock will be assigned if the static detection fails:

<table>
<thead>
<tr>
<th>Query</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db:get(doc('test')/reference/text())</td>
<td>The name of the database to be opened will only be known at evaluation time.</td>
</tr>
<tr>
<td>(1 to 100) ! db:get(concat('db', .))</td>
<td>The UNROLLLIMIT can be increased to generate 100 db:get function calls and corresponding locks.</td>
</tr>
</tbody>
</table>

The functions fn:doc and fn:collection can be used for both accessing databases resources and fetching resources at the specified URI (see Access Resources for more details). There are two ways to reduce the number of locks:

1. Turn off the WITHDB option to prevent the functions from accessing databases; or
2. use fetch:doc for fetching resources from URIs, and use db:get for accessing databases.

You can consult the query info output (via the Info View of the GUI, via -V on Command-Line or via turning on the QUERYINFO option) to find out which databases are locked by a query, and if local locks or a global lock is applied.

XQuery Locks

By default, access to external resources (files on hard disk, HTTP requests, …) is not controlled by the transaction monitor of BaseX. Custom locks can be assigned via annotations, pragmas or options:
• A lock string may consist of a single key or multiple keys separated with commas.

• Internal locks and XQuery locks can co-exist. No conflicts arise, even if a lock string equals the name of a database that is locked by the transaction manager.

• The lock is transformed into a write lock by making the corresponding expression updating.

Annotations

In the following module, lock annotations are used to prevent concurrent write operations on the same file:

```xml
module namespace config = 'config';

declare %base:lock('CONFIG') function config:read() as xs:string {
  file:read-text('config.txt')
};

declare %updating %base:lock('CONFIG') function config:write($data as xs:string) {
  file:write-text('config.txt', $data)
};
```

Some explanations:

• If a query calls the `config:read` function, a read lock will be acquired for the user-defined `CONFIG` lock string before query evaluation.

• If `config:write` is called by a query, a write lock will be applied.

• If another query calls `config:write`, it will be queued until the first query is evaluated.

Pragmas

Locks can also be declared via pragmas:

```xml
update:output((# basex:lock CONFIG #) {
  file:write('config.xml', <config/>)
})
```

The write locks is enforced via the Update.

Options

Locks for the functions of a module can also be assigned via option declarations:

```xml
declare option basex:lock 'CONFIG';
update:output(file:write('config.xml', <config/>))
```

Once again, a write lock is enforced.

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.

File-System Locks

Update Operations

During 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 will not be deleted. In this case, the
database cannot be opened anymore, 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 that are caused by accidental write operations from 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, and if no exclusive lock can be acquired, it will be rejected with the message "Database ... is currently opened by another process.”.

Please note that you cannot 100% rely on this mechanism, as it is not possible to synchronize operations across different JVMs. You will be safe when using the client/server or HTTP architecture.

Changelog

Version 10.0
• Updated: Lock detection was improved by splitting compilation into multiple steps.

Version 9.4
• Updated: Single lock option for reads and writes.

Version 9.1
• Updated: Query lock options were moved from query to basex namespace.

Version 8.6
• Updated: New FAIRLOCK option, improved detection of lock patterns.

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 108. 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.

Introduction

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 beginning with a . may be hidden. The log directory can be changed via the LOGPATH option.

If BaseX is used in a Web Application, all trace output (generated via fn:trace, prof:dump and similar functions) will be stored in the logs as well.

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.

If a proxy is used, the original IP address of the client will be added to the logs.

RESTXQ

By default, RESTXQ code is executed with the admin user. As a result, this user will be displayed in the logs for all RESTXQ requests. In a web application with a custom user management, however, the name of the actual user who has sent a request is often more relevant.

When log data is written during the processing of a RESTXQ function, the following is looked up as follows:

1. The current request is checked for an id attribute. The attribute can be assigned via RESTXQ and the request:set-attribute function, and it is the recommended approach for stateless requests as all request attributes will be dropped after the finalization of a request.
2. If none is found, the id attribute is looked up in the current user session. The attribute can be assigned via session:set (see e.g. the DBA code for sessions and user handling). If the request path contains a dba segment, a dba session attribute will be looked up instead.
3. If none is found, the default path will be taken, and the user of the current database context will be included in the logs.

Format

Example 1

<table>
<thead>
<tr>
<th>Time</th>
<th>IP Address</th>
<th>User</th>
<th>Result</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:18:12.892</td>
<td>SERVER 127.0.0.1:4722</td>
<td>admin</td>
<td>OK</td>
<td>Server was started (port: 1984)</td>
</tr>
<tr>
<td>01:18:15.436</td>
<td>random:double()</td>
<td>jack</td>
<td>REQUEST</td>
<td>XQUERY for $i in 1 to 5 return random:double()</td>
</tr>
<tr>
<td>01:18:15.446</td>
<td>127.0.0.1:4722</td>
<td>jack</td>
<td>OK</td>
<td>Query executed in 2.38 ms.</td>
</tr>
<tr>
<td>01:18:15.447</td>
<td>127.0.0.1:4722</td>
<td>2.72 ms</td>
<td>OK</td>
<td>Query executed in 2.38 ms.</td>
</tr>
<tr>
<td>01:18:15.447</td>
<td>127.0.0.1:4722</td>
<td>jack</td>
<td>REQUEST</td>
<td>EXIT</td>
</tr>
<tr>
<td>01:18:15.447</td>
<td>127.0.0.1:4722</td>
<td>jack</td>
<td>OK</td>
<td>Query executed in 0.39 ms.</td>
</tr>
</tbody>
</table>
A server has been started and a user jack has connected to the server to perform a query and exit properly.

Example 2

<table>
<thead>
<tr>
<th>Time</th>
<th>Address</th>
<th>User</th>
<th>Status</th>
<th>Event</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:23:33.251</td>
<td>127.0.0.1:4736</td>
<td>john</td>
<td>OK</td>
<td>QUERY</td>
<td>'hi'</td>
</tr>
<tr>
<td>01:23:33.337</td>
<td>127.0.0.1:4736</td>
<td>john</td>
<td>OK</td>
<td>ITER</td>
<td></td>
</tr>
<tr>
<td>01:23:33.338</td>
<td>127.0.0.1:4736</td>
<td>john</td>
<td>OK</td>
<td>INFO</td>
<td></td>
</tr>
<tr>
<td>01:23:33.339</td>
<td>127.0.0.1:4736</td>
<td>john</td>
<td>OK</td>
<td>CLOSE</td>
<td></td>
</tr>
<tr>
<td>01:23:33.359</td>
<td>127.0.0.1:4736</td>
<td>john</td>
<td>REQUEST</td>
<td>EXIT</td>
<td></td>
</tr>
</tbody>
</table>

A user john has performed an iterative query, using one of the client APIs.

Example 3

<table>
<thead>
<tr>
<th>Time</th>
<th>Address</th>
<th>User</th>
<th>Status</th>
<th>Event</th>
<th>URL</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:31:51.892</td>
<td>127.0.0.1:4803</td>
<td>admin</td>
<td>200</td>
<td></td>
<td></td>
<td>4.43 ms</td>
</tr>
</tbody>
</table>

An admin user has accessed the factbook database via REST.

**Changelog**

Version 9.5

- Updated: Show IP address behind proxy.

Version 9.3

- Updated: Store trace output in database logs
- Updated: RESTXQ: The request attributes will be checked for a user id.

Version 8.6

- Added: The log directory can be changed with the LOGPATH option.
- Updated: Include session attributes in log data.
Part XIII. Use Cases
## Chapter 109. Statistics

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It lists statistics on various databases instances that have been created with BaseX, with value and full-text indexes turned off. The URLs to the original sources, if available or public, are listed below.

Databases in BaseX are light-weight. If a database limit is reached, you can distribute your documents across multiple database instances and access all of them with a single XQuery expression.

### Databases

<table>
<thead>
<tr>
<th>Instances</th>
<th>FileSize</th>
<th>#Files</th>
<th>DbSize</th>
<th>#Nodes</th>
<th>#Attr</th>
<th>#ENames</th>
<th>#ANames</th>
<th>#URIs</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limits</td>
<td>512 GiB</td>
<td>536'870'912 MiBi (2^39 Bytes)</td>
<td>2'147'483'648 MiBi (2^31)</td>
<td>32768 (2^15)</td>
<td>2^256 (2^28)</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RuWikiHist</td>
<td>421 GiB</td>
<td>1</td>
<td>416 GiB</td>
<td>324'848'508 MiBi</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>ZhWikiHist</td>
<td>26 GiB</td>
<td>1</td>
<td>120 GiB</td>
<td>179'199'662 MiBi</td>
<td>21</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>EnWikt</td>
<td>25 GiB</td>
<td>1</td>
<td>75 GiB</td>
<td>134'380'393 MiBi</td>
<td>21</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td>55 GiB</td>
<td>1</td>
<td>64 GiB</td>
<td>1'615'071'348 MiBi</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>EnWikiMeta</td>
<td>26 GiB</td>
<td>1</td>
<td>25 GiB</td>
<td>198'546'747 MiBi</td>
<td>24</td>
<td>21</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>MedLine</td>
<td>38 GiB</td>
<td>1</td>
<td>52 GiB</td>
<td>401'456'348 MiBi</td>
<td>21</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>iProClass</td>
<td>36 GiB</td>
<td>1</td>
<td>37 GiB</td>
<td>1'631'218'984 MiBi</td>
<td>245</td>
<td>4</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Inex2009</td>
<td>31 GiB</td>
<td>1</td>
<td>266'650 GiBi</td>
<td>1'336'110'639 MiBi</td>
<td>280'034</td>
<td>451</td>
<td>1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>CoPhIR</td>
<td>29 GiB</td>
<td>10'000'000 31 GiBi</td>
<td>1'104'623'376 MiBi</td>
<td>101</td>
<td>2</td>
<td>6</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnWiki</td>
<td>55 GiB</td>
<td>1</td>
<td>64 GiB</td>
<td>1'615'071'348 MiBi</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>InterPro</td>
<td>14 GiB</td>
<td>1</td>
<td>19 GiB</td>
<td>860'304'215 MiBi</td>
<td>7</td>
<td>15</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Genome1</td>
<td>13 GiB</td>
<td>1</td>
<td>13 GiB</td>
<td>432'628'103 MiBi</td>
<td>26</td>
<td>101</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>NewYork</td>
<td>14 GiB</td>
<td>1</td>
<td>19 GiB</td>
<td>860'304'215 MiBi</td>
<td>7</td>
<td>15</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>TrEMBL</td>
<td>11 GiB</td>
<td>1</td>
<td>14 GiB</td>
<td>589'650'538 MiBi</td>
<td>47</td>
<td>30</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td>11 GiB</td>
<td>1</td>
<td>13 GiB</td>
<td>323'083'402 MiBi</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>IntAct</td>
<td>7973 MiBi</td>
<td>25'624</td>
<td>6717 MiBi</td>
<td>297'478'392 MiBi</td>
<td>64</td>
<td>22</td>
<td>2</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Freebase</td>
<td>7366 MiBi</td>
<td>1</td>
<td>10 GiB</td>
<td>443'627'998 MiBi</td>
<td>61</td>
<td>283</td>
<td>1</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>SDMX</td>
<td>6356 MiBi</td>
<td>1</td>
<td>8028 MiBi</td>
<td>395'871'872 MiBi</td>
<td>22</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>OpenStreet</td>
<td>64 MiBi</td>
<td>1</td>
<td>571 MiBi</td>
<td>6'910'669 MiBi</td>
<td>19</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SwissProt</td>
<td>4604 MiBi</td>
<td>1</td>
<td>5422 MiBi</td>
<td>241'274'408 MiBi</td>
<td>70</td>
<td>39</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>EURLex</td>
<td>4815 MiBi</td>
<td>1</td>
<td>5532 MiBi</td>
<td>167'328'039 MiBi</td>
<td>186</td>
<td>46</td>
<td>1</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Wikicorpus</td>
<td>4492 MiBi</td>
<td>659'338</td>
<td>4432 MiBi</td>
<td>157'948'562 MiBi</td>
<td>1'257</td>
<td>2'687</td>
<td>2</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>EnWikiRDF</td>
<td>8679 MiBi</td>
<td>1</td>
<td>3537 MiBi</td>
<td>98'433'194 MiBi</td>
<td>11</td>
<td>2</td>
<td>11</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>CoPhIR</td>
<td>2695 MiBi</td>
<td>150'000</td>
<td>2882 MiBi</td>
<td>101'638'851 MiBi</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>MeSH</td>
<td>2091 MiBi</td>
<td>1</td>
<td>2410 MiBi</td>
<td>104'845'819 MiBi</td>
<td>6</td>
<td>5</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>FreeDB</td>
<td>1723 MiBi</td>
<td>1</td>
<td>2462 MiBi</td>
<td>102'901'512 MiBi</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td>1134 MiBi</td>
<td>1</td>
<td>1303 MiBi</td>
<td>32'298'989 MiBi</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Library</td>
<td>Size</td>
<td>Files</td>
<td>#Files</td>
<td>#DbSize</td>
<td>#Nodes</td>
<td>#Attr</td>
<td>#ENames</td>
<td>#ANames</td>
<td>#URIs</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>DeepFS</td>
<td>810 MiB</td>
<td>1</td>
<td>850 MiB</td>
<td>44'821'5064</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>LibraryUK</td>
<td>560 MiB</td>
<td>1</td>
<td>918 MiB</td>
<td>46'401'9413</td>
<td>23</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Twitter</td>
<td>736 MiB</td>
<td>1</td>
<td>1'777'945</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>1'019'132</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>DBLP</td>
<td>694 MiB</td>
<td>1</td>
<td>944 MiB</td>
<td>36'878'1814</td>
<td>35</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Feeds</td>
<td>692 MiB</td>
<td>1</td>
<td>444'014</td>
<td>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>MedLineS</td>
<td>447 MiB</td>
<td>1</td>
<td>407 MiB</td>
<td>21'602'1415</td>
<td>55</td>
<td>7</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>AirBase</td>
<td>449 MiB</td>
<td>38</td>
<td>273 MiB</td>
<td>14'512'8511</td>
<td>111</td>
<td>5</td>
<td>0</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>MedLineDB</td>
<td>260 MiB</td>
<td>1</td>
<td>195 MiB</td>
<td>10'401'8475</td>
<td>66</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>ZDNET</td>
<td>130 MiB</td>
<td>95'663</td>
<td>133 MiB</td>
<td>3'060'186</td>
<td>40</td>
<td>90</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>JMNEdict</td>
<td>124 MiB</td>
<td>1</td>
<td>171 MiB</td>
<td>8'592'666</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td>111 MiB</td>
<td>1</td>
<td>130 MiB</td>
<td>3'221'926</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Freshmeat</td>
<td>105 MiB</td>
<td>1</td>
<td>86 MiB</td>
<td>3'832'028</td>
<td>58</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>DeepFS</td>
<td>83 MiB</td>
<td>1</td>
<td>93 MiB</td>
<td>4'842'638</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Treebank</td>
<td>82 MiB</td>
<td>1</td>
<td>92 MiB</td>
<td>3'829'513</td>
<td>250</td>
<td>1</td>
<td>0</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>DBLP2</td>
<td>80 MiB</td>
<td>1</td>
<td>170'843</td>
<td>102 MiB</td>
<td>4'044'649</td>
<td>35</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>DDI</td>
<td>76 MiB</td>
<td>3</td>
<td>39 MiB</td>
<td>2'070'157</td>
<td>104</td>
<td>16</td>
<td>21</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Alfred</td>
<td>75 MiB</td>
<td>1</td>
<td>68 MiB</td>
<td>3'784'285</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>University</td>
<td>56 MiB</td>
<td>6</td>
<td>66 MiB</td>
<td>3'468'606</td>
<td>28</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>MediaUK</td>
<td>38 MiB</td>
<td>1</td>
<td>45 MiB</td>
<td>1'619'443</td>
<td>21</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>HCIBIB2</td>
<td>32 MiB</td>
<td>1</td>
<td>33 MiB</td>
<td>617'023</td>
<td>39</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>MediaUK</td>
<td>24 MiB</td>
<td>1</td>
<td>25 MiB</td>
<td>845'805</td>
<td>61</td>
<td>8</td>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Nasa</td>
<td>2292 KiB</td>
<td>1</td>
<td>2359 KiB</td>
<td>80'178</td>
<td>54</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Shakespeare</td>
<td>7711 KiB</td>
<td>1</td>
<td>9854 KiB</td>
<td>327'170</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>TreeOfLife</td>
<td>5425 KiB</td>
<td>1</td>
<td>363560 KiB</td>
<td>7</td>
<td>4</td>
<td>7</td>
<td>0</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>Thesaurus</td>
<td>4288 KiB</td>
<td>1</td>
<td>4088 KiB</td>
<td>201'798</td>
<td>33</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>MusicXML</td>
<td>3155 KiB</td>
<td>17</td>
<td>2942 KiB</td>
<td>171'400</td>
<td>179</td>
<td>56</td>
<td>0</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>BibDBPub</td>
<td>2292 KiB</td>
<td>3'465</td>
<td>2359 KiB</td>
<td>80'178</td>
<td>54</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Factbook</td>
<td>1743 KiB</td>
<td>1</td>
<td>1560 KiB</td>
<td>77'315</td>
<td>23</td>
<td>32</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td>1134 KiB</td>
<td>1</td>
<td>1334 KiB</td>
<td>33'056</td>
<td>74</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

This is the meaning of the attributes:

- **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

## Sources

<table>
<thead>
<tr>
<th>Instances</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>AirBase</td>
<td><a href="http://air-climate.eionet.europa.eu/databases/airbase/airbase.xml">http://air-climate.eionet.europa.eu/databases/airbase/airbase.xml</a></td>
</tr>
<tr>
<td>Alfred</td>
<td><a href="http://alfred.med.yale.edu/alfred/alfredWithDescription.zip">http://alfred.med.yale.edu/alfred/alfredWithDescription.zip</a></td>
</tr>
<tr>
<td>BibDBPub</td>
<td><a href="http://inex.is.informatik.uni-duisburg.de/2005/">http://inex.is.informatik.uni-duisburg.de/2005/</a></td>
</tr>
<tr>
<td>CoPhIR</td>
<td><a href="http://cophir.isti.cnr.it/">http://cophir.isti.cnr.it/</a></td>
</tr>
<tr>
<td>DBLP</td>
<td><a href="http://dblp.uni-trier.de/xml">http://dblp.uni-trier.de/xml</a></td>
</tr>
<tr>
<td>DBLP2</td>
<td><a href="http://inex.is.informatik.uni-duisburg.de/2005/">http://inex.is.informatik.uni-duisburg.de/2005/</a></td>
</tr>
<tr>
<td>DDI</td>
<td><a href="http://tools.ddialliance.org/">http://tools.ddialliance.org/</a></td>
</tr>
<tr>
<td>EnWikiRDF</td>
<td><a href="http://www.xml-benchmark.org/generated">http://www.xml-benchmark.org/generated</a> with xmlgen</td>
</tr>
<tr>
<td>EnWiktionary</td>
<td><a href="http://dumps.wikimedia.org/enwiktionary/latest/enwiktionary-latest-pages-meta-history.xml.7z">http://dumps.wikimedia.org/enwiktionary/latest/enwiktionary-latest-pages-meta-history.xml.7z</a></td>
</tr>
<tr>
<td>EURLex</td>
<td><a href="http://www.epsiplatform.eu/">http://www.epsiplatform.eu/</a></td>
</tr>
<tr>
<td>Factbook</td>
<td><a href="http://www.cs.washington.edu/research/xmldatasets/www/repository.html">http://www.cs.washington.edu/research/xmldatasets/www/repository.html</a></td>
</tr>
<tr>
<td>Freebase</td>
<td><a href="http://download.freebase.com/wex">http://download.freebase.com/wex</a></td>
</tr>
<tr>
<td>FreeDB</td>
<td><a href="http://www.xmldatabases.org/radio/xmlDatabases/projects/FreeDBtoXML">http://www.xmldatabases.org/radio/xmlDatabases/projects/FreeDBtoXML</a></td>
</tr>
<tr>
<td>Freshmeat</td>
<td><a href="http://freshmeat.net/articles/freshmeat-xml-rpc-api-available">http://freshmeat.net/articles/freshmeat-xml-rpc-api-available</a></td>
</tr>
<tr>
<td>HCIBIB2</td>
<td><a href="http://inex.is.informatik.uni-duisburg.de/2005/">http://inex.is.informatik.uni-duisburg.de/2005/</a></td>
</tr>
<tr>
<td>InterPro</td>
<td>ftp://ftp.bio.net/biomirror/interpro/match_complete.xml.gz</td>
</tr>
<tr>
<td>iProClass</td>
<td>ftp://ftp.pir.georgetown.edu/pir_databases/iproclass/iproclass.xml.gz</td>
</tr>
<tr>
<td>KanjiDic2</td>
<td><a href="http://www.csse.monash.edu.au/~jwb/kanjidic2">http://www.csse.monash.edu.au/~jwb/kanjidic2</a></td>
</tr>
<tr>
<td>MovieDB</td>
<td><a href="http://eagereyes.org/InfoVisContest2007Data.html">http://eagereyes.org/InfoVisContest2007Data.html</a></td>
</tr>
<tr>
<td>Dataset</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>MusicXML</td>
<td></td>
</tr>
<tr>
<td>Nasa</td>
<td></td>
</tr>
<tr>
<td>OpenStreetMap</td>
<td></td>
</tr>
<tr>
<td>Organizations</td>
<td></td>
</tr>
<tr>
<td>RuWikiHist</td>
<td></td>
</tr>
<tr>
<td>SDMX</td>
<td></td>
</tr>
<tr>
<td>Shakespeare</td>
<td></td>
</tr>
<tr>
<td>Thesaurus</td>
<td></td>
</tr>
<tr>
<td>Treebank</td>
<td></td>
</tr>
<tr>
<td>TreeOfLife</td>
<td></td>
</tr>
<tr>
<td>Wikicorpus</td>
<td></td>
</tr>
<tr>
<td>XMark</td>
<td></td>
</tr>
<tr>
<td>ZDNET</td>
<td></td>
</tr>
<tr>
<td>ZhWikiHist</td>
<td></td>
</tr>
<tr>
<td>LibraryUKN</td>
<td>generated from university library data</td>
</tr>
<tr>
<td>MediaUKN</td>
<td>generated from university library data</td>
</tr>
<tr>
<td>DeepFS</td>
<td>generated from filesystem structure</td>
</tr>
<tr>
<td>University</td>
<td>generated from students test data</td>
</tr>
<tr>
<td>Feeds</td>
<td>compiled from news feeds</td>
</tr>
<tr>
<td>Twitter</td>
<td>compiled from Twitter feeds</td>
</tr>
</tbody>
</table>
Chapter 110. Twitter

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 data is 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

<table>
<thead>
<tr>
<th>Day</th>
<th>Description</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon, 6-Feb-2012</td>
<td>Total tweets</td>
<td>30.824.976</td>
</tr>
<tr>
<td></td>
<td>Average tweets per hour</td>
<td>1.284.374</td>
</tr>
<tr>
<td></td>
<td>Average tweets per minute</td>
<td>21.406</td>
</tr>
<tr>
<td></td>
<td>Average tweets per second</td>
<td>356</td>
</tr>
<tr>
<td>Tue, 6-Mar-2012</td>
<td>Total tweets</td>
<td>31.823.776</td>
</tr>
<tr>
<td></td>
<td>Average tweets per hour</td>
<td>1.325.990</td>
</tr>
<tr>
<td></td>
<td>Average tweets per minute</td>
<td>22.099</td>
</tr>
<tr>
<td></td>
<td>Average tweets per second</td>
<td>368</td>
</tr>
<tr>
<td>Date</td>
<td>Total tweets</td>
<td>Average tweets per hour</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Fri, 6-Apr-2012</td>
<td>34,638,976</td>
<td>1,443.290</td>
</tr>
<tr>
<td>Sun, 6-May-2012</td>
<td>35,982,976</td>
<td>1,499.290</td>
</tr>
</tbody>
</table>

**Example Tweet (JSON)**

```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": "&lt;a href="http://twitterfeed.com" rel="nofollow\n\&gt;twitterfeed&lt;/a&gt;",
  "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": "CODEED",
    "name": "BaseX",
    "profile_background_color": "CODEED",
    "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 favored"
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/>
</entities>
</text>
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.

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

<table>
<thead>
<tr>
<th>Amount of tweets</th>
<th>Time in seconds</th>
<th>Time in minutes</th>
<th>Database Size (without indexes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000.000</td>
<td>492.26346</td>
<td>8.2</td>
<td>3396 MB</td>
</tr>
<tr>
<td>2.000.000</td>
<td>461.87326</td>
<td>7.6</td>
<td>6997 MB</td>
</tr>
<tr>
<td>3.000.000</td>
<td>470.7054</td>
<td>7.8</td>
<td>10452 MB</td>
</tr>
</tbody>
</table>

![Performance of BaseX - Inserting tweets into DB](image-url)