<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Scripts</td>
<td>48</td>
</tr>
<tr>
<td>Glob Syntax</td>
<td>48</td>
</tr>
<tr>
<td>Valid Names</td>
<td>49</td>
</tr>
<tr>
<td>Aliases</td>
<td>49</td>
</tr>
<tr>
<td>Database Operations</td>
<td>49</td>
</tr>
<tr>
<td>CREATE DB</td>
<td>49</td>
</tr>
<tr>
<td>OPEN</td>
<td>49</td>
</tr>
<tr>
<td>CHECK</td>
<td>50</td>
</tr>
<tr>
<td>CLOSE</td>
<td>50</td>
</tr>
<tr>
<td>EXPORT</td>
<td>50</td>
</tr>
<tr>
<td>CREATE INDEX</td>
<td>50</td>
</tr>
<tr>
<td>DROP INDEX</td>
<td>50</td>
</tr>
<tr>
<td>Administration</td>
<td>51</td>
</tr>
<tr>
<td>ALTER DB</td>
<td>51</td>
</tr>
<tr>
<td>DROP DB</td>
<td>51</td>
</tr>
<tr>
<td>CREATE BACKUP</td>
<td>51</td>
</tr>
<tr>
<td>RESTORE</td>
<td>51</td>
</tr>
<tr>
<td>INSPECT</td>
<td>52</td>
</tr>
<tr>
<td>DROP BACKUP</td>
<td>52</td>
</tr>
<tr>
<td>SHOW BACKUPS</td>
<td>52</td>
</tr>
<tr>
<td>COPY</td>
<td>52</td>
</tr>
<tr>
<td>INFO DB</td>
<td>52</td>
</tr>
<tr>
<td>INFO INDEX</td>
<td>52</td>
</tr>
<tr>
<td>INFO STORAGE</td>
<td>53</td>
</tr>
<tr>
<td>Querying</td>
<td>53</td>
</tr>
<tr>
<td>LIST</td>
<td>53</td>
</tr>
<tr>
<td>XQUERY</td>
<td>53</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>53</td>
</tr>
<tr>
<td>FIND</td>
<td>54</td>
</tr>
<tr>
<td>TEST</td>
<td>54</td>
</tr>
<tr>
<td>REPO INSTALL</td>
<td>54</td>
</tr>
<tr>
<td>REPO LIST</td>
<td>54</td>
</tr>
<tr>
<td>REPO DELETE</td>
<td>54</td>
</tr>
<tr>
<td>Updates</td>
<td>55</td>
</tr>
<tr>
<td>ADD</td>
<td>55</td>
</tr>
<tr>
<td>DELETE</td>
<td>55</td>
</tr>
<tr>
<td>RENAME</td>
<td>55</td>
</tr>
<tr>
<td>REPLACE</td>
<td>56</td>
</tr>
<tr>
<td>STORE</td>
<td>56</td>
</tr>
<tr>
<td>OPTIMIZE</td>
<td>56</td>
</tr>
<tr>
<td>FLUSH</td>
<td>56</td>
</tr>
<tr>
<td>Monitoring</td>
<td>57</td>
</tr>
<tr>
<td>SHOW SESSIONS</td>
<td>57</td>
</tr>
<tr>
<td>SHOW USERS</td>
<td>57</td>
</tr>
<tr>
<td>KILL</td>
<td>57</td>
</tr>
<tr>
<td>JOBS LIST</td>
<td>57</td>
</tr>
<tr>
<td>JOBS RESULT</td>
<td>57</td>
</tr>
<tr>
<td>JOBS STOP</td>
<td>58</td>
</tr>
<tr>
<td>User Management</td>
<td>58</td>
</tr>
<tr>
<td>CREATE USER</td>
<td>58</td>
</tr>
<tr>
<td>ALTER USER</td>
<td>58</td>
</tr>
<tr>
<td>ALTER PASSWORD</td>
<td>58</td>
</tr>
<tr>
<td>DROP USER</td>
<td>58</td>
</tr>
<tr>
<td>GRANT</td>
<td>59</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>59</td>
</tr>
<tr>
<td>General Commands</td>
<td>59</td>
</tr>
<tr>
<td>RUN</td>
<td>59</td>
</tr>
<tr>
<td>EXECUTE</td>
<td>59</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>19</td>
<td>Integrating Eclipse</td>
</tr>
<tr>
<td></td>
<td>Integration</td>
</tr>
<tr>
<td></td>
<td>Windows</td>
</tr>
<tr>
<td></td>
<td>Linux</td>
</tr>
<tr>
<td></td>
<td>Mac OSX</td>
</tr>
<tr>
<td></td>
<td>Setting up</td>
</tr>
<tr>
<td></td>
<td>Setting up as Standalone</td>
</tr>
<tr>
<td></td>
<td>Setting up as Client</td>
</tr>
<tr>
<td></td>
<td>Usage</td>
</tr>
<tr>
<td>V</td>
<td>Query Features</td>
</tr>
<tr>
<td>20</td>
<td>XQuery</td>
</tr>
<tr>
<td>21</td>
<td>XQuery 3.0</td>
</tr>
<tr>
<td></td>
<td>Enhanced FLWOR Expressions</td>
</tr>
<tr>
<td></td>
<td>group by</td>
</tr>
<tr>
<td></td>
<td>count</td>
</tr>
<tr>
<td></td>
<td>allowing empty</td>
</tr>
<tr>
<td></td>
<td>window</td>
</tr>
<tr>
<td></td>
<td>Function Items</td>
</tr>
<tr>
<td></td>
<td>Simple Map Operator</td>
</tr>
<tr>
<td></td>
<td>Try/Catch</td>
</tr>
<tr>
<td></td>
<td>Switch</td>
</tr>
<tr>
<td></td>
<td>Expanded QNames</td>
</tr>
<tr>
<td></td>
<td>Namespace Constructors</td>
</tr>
<tr>
<td></td>
<td>String Concatenations</td>
</tr>
<tr>
<td></td>
<td>External Variables</td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
</tr>
<tr>
<td></td>
<td>Context Item</td>
</tr>
<tr>
<td></td>
<td>Annotations</td>
</tr>
<tr>
<td></td>
<td>Functions</td>
</tr>
<tr>
<td></td>
<td>Changelog</td>
</tr>
<tr>
<td>22</td>
<td>Higher-Order Functions</td>
</tr>
<tr>
<td></td>
<td>Function Items</td>
</tr>
<tr>
<td></td>
<td>Function Types</td>
</tr>
<tr>
<td></td>
<td>Higher-Order Functions</td>
</tr>
<tr>
<td></td>
<td>Sequences</td>
</tr>
<tr>
<td></td>
<td>Folds</td>
</tr>
<tr>
<td>23</td>
<td>XQuery 3.1</td>
</tr>
<tr>
<td></td>
<td>Maps</td>
</tr>
<tr>
<td></td>
<td>Arrays</td>
</tr>
<tr>
<td></td>
<td>Atomization</td>
</tr>
<tr>
<td></td>
<td>Lookup Operator</td>
</tr>
<tr>
<td></td>
<td>Arrow Operator</td>
</tr>
<tr>
<td></td>
<td>String Constructor</td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
</tr>
<tr>
<td></td>
<td>Adaptive Serialization</td>
</tr>
<tr>
<td></td>
<td>JSON Serialization</td>
</tr>
<tr>
<td></td>
<td>Functions</td>
</tr>
<tr>
<td></td>
<td>Map Functions</td>
</tr>
<tr>
<td></td>
<td>Array Functions</td>
</tr>
<tr>
<td></td>
<td>JSON Functions</td>
</tr>
<tr>
<td></td>
<td>fn:sort</td>
</tr>
<tr>
<td></td>
<td>fn:contains-token</td>
</tr>
<tr>
<td></td>
<td>fn:parse-ietf-date</td>
</tr>
<tr>
<td></td>
<td>fn:apply</td>
</tr>
<tr>
<td></td>
<td>fn:random-number-generator</td>
</tr>
</tbody>
</table>
Java ................................................................. 134
Performance .................................................. 135
Changelog ................................................. 135
28. Full-Text .................................................. 136
   Introduction ............................................. 136
   Combining Results ..................................... 136
   Positional Filters ..................................... 137
   Match Options ........................................ 137
BaseX Features ............................................. 138
   Languages .............................................. 138
   Scoring ............................................... 139
   Thesaurus ............................................. 139
   Fuzzy Querying ...................................... 140
Mixed Content ............................................. 140
Functions .................................................. 141
Collations .................................................. 141
Changelog ................................................. 142
29. Full-Text: Japanese .................................... 143
   Introduction .......................................... 143
   Lexical Analysis ...................................... 143
   Parsing ............................................... 143
   Token Processing .................................... 144
   Stemming ............................................. 144
   Wildcards ............................................ 144
30. XQuery Update ......................................... 145
   Features ............................................... 145
      Updating Expressions ......................... 145
      Non-Updating Expressions ................ 146
      Functions ....................................... 147
   Concepts ............................................. 148
      Pending Update List ......................... 148
      Returning Results .............................. 149
      Effects .......................................... 149
      Original Files ................................... 149
      Indexes .......................................... 149
   Error Messages ..................................... 149
   Changelog .......................................... 149
31. Indexes .................................................. 151
   Structural Indexes ................................ 151
      Name Index ...................................... 151
      Path Index ...................................... 151
      Document Index ................................ 152
   Value Indexes ...................................... 152
      Text Index ...................................... 152
      Attribute Index ................................. 153
      Token Index .................................... 153
      Full-Text Index ................................. 154
      Selective Indexing ............................ 154
      Enforce Rewritings ......................... 155
   Custom Index Structures ......................... 156
Performance ........................................... 156
Updates .................................................. 157
Changelog ................................................. 157
32. Serialization ........................................... 159
   Parameters .......................................... 159
   Character mappings ................................ 162
   Changelog ........................................... 162
33. XQuery Errors ......................................... 164
VI. XQuery Modules .................................................................................. 173

34. Admin Module .................................................................................. 174
   Conventions ...................................................................................... 174
   Functions ......................................................................................... 174
       admin:sessions ........................................................................... 174
       admin:logs .................................................................................. 174
       admin:write-log ......................................................................... 174
       admin:delete-logs ...................................................................... 174
   Errors ............................................................................................... 175
   Changelog ....................................................................................... 175

35. Archive Module .............................................................................. 176
   Conventions ...................................................................................... 176
   Functions ......................................................................................... 176
       archive:create ........................................................................... 176
       archive:create-from ............................................................... 177
       archive:entries .......................................................................... 177
       archive:options ......................................................................... 177
       archive:extract-text ............................................................... 178
       archive:extract-binary ............................................................ 178
       archive:extract-to .................................................................... 178
       archive:update ........................................................................... 179
       archive:delete ........................................................................... 179
   Errors ............................................................................................... 179
   Changelog ....................................................................................... 179

36. Array Module .................................................................................. 181
   Conventions ...................................................................................... 181
   Functions ......................................................................................... 181
       array:size .................................................................................. 181
       array:get .................................................................................... 181
       array:append ............................................................................. 181
       array:subarray .......................................................................... 181
       array:put .................................................................................... 181
       array:remove ............................................................................. 182
       array:insert-before ................................................................... 182
       array:head ................................................................................ 182
       array:tail ................................................................................... 182
       array:reverse ............................................................................. 182
       array:join .................................................................................. 182
       array:flatten .............................................................................. 183
       array:for-each .......................................................................... 183
       array:filter ................................................................................ 183
       array:fold-left .......................................................................... 183
       array:fold-right ........................................................................ 184
       array:for-each-pair ................................................................... 184
       array:sort .................................................................................. 184
   Errors ............................................................................................... 184
   Changelog ....................................................................................... 185

37. Binary Module ............................................................................... 186
   Conventions ...................................................................................... 186
   Constants and Conversions ............................................................ 186
38. Client Module

Conventions ................................................. 192

Functions
   client:connect ........................................ 192
   client:execute ........................................ 192
   client:info ........................................... 192
   client:query .......................................... 193
   client:close .......................................... 193

Errors ..................................................... 193

Changelog ................................................. 194

39. Conversion Module

Conventions ................................................. 195

Strings .................................................... 195
   convert:binary-to-string ........................... 195
   convert:string-to-base64 ......................... 195
   convert:string-to-hex ............................. 195

Binary Data .............................................. 196
   convert:integer-to-string ....................... 196
   convert:integer-to-base ......................... 196
   convert:integer-from-base ...................... 196
   convert:integer-to-integers ................... 196
   convert:integer-to-bytes ....................... 196

Numbers .................................................. 196
   convert:integer-to-base ......................... 196
   convert:integer-from-base ...................... 197

Dates and Durations .................................... 197
   convert:integer-to-dateTime .................... 197

Errors ..................................................... 198

Changelog ................................................. 199
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Module</td>
<td>209</td>
</tr>
<tr>
<td>Cryptographic Module</td>
<td>199</td>
</tr>
<tr>
<td>CSV Module</td>
<td>204</td>
</tr>
<tr>
<td>Database Module</td>
<td>209</td>
</tr>
<tr>
<td>Conventions</td>
<td></td>
</tr>
<tr>
<td>Conversion</td>
<td>204</td>
</tr>
<tr>
<td>Options</td>
<td>205</td>
</tr>
<tr>
<td>Functions</td>
<td>206</td>
</tr>
<tr>
<td>Functions (csv:parse, csv:serialize)</td>
<td>206</td>
</tr>
<tr>
<td>Examples</td>
<td>206</td>
</tr>
<tr>
<td>Errors</td>
<td>207</td>
</tr>
<tr>
<td>Changelog</td>
<td>203</td>
</tr>
<tr>
<td>40. Cryptographic Module</td>
<td></td>
</tr>
<tr>
<td>Conventions</td>
<td>199</td>
</tr>
<tr>
<td>Message Authentication</td>
<td>199</td>
</tr>
<tr>
<td>crypto:hmac</td>
<td>199</td>
</tr>
<tr>
<td>Encryption &amp; Decryption</td>
<td>200</td>
</tr>
<tr>
<td>crypto:encrypt</td>
<td>200</td>
</tr>
<tr>
<td>crypto:decrypt</td>
<td>200</td>
</tr>
<tr>
<td>XML Signatures</td>
<td>201</td>
</tr>
<tr>
<td>crypto:generate-signature</td>
<td>201</td>
</tr>
<tr>
<td>crypto:validate-signature</td>
<td>202</td>
</tr>
<tr>
<td>Errors</td>
<td>203</td>
</tr>
<tr>
<td>Changelog</td>
<td>203</td>
</tr>
<tr>
<td>41. CSV Module</td>
<td></td>
</tr>
<tr>
<td>Conventions</td>
<td>204</td>
</tr>
<tr>
<td>Conversion</td>
<td>204</td>
</tr>
<tr>
<td>Options</td>
<td>205</td>
</tr>
<tr>
<td>Functions</td>
<td>206</td>
</tr>
<tr>
<td>csv:parse</td>
<td>206</td>
</tr>
<tr>
<td>csv:serialize</td>
<td>206</td>
</tr>
<tr>
<td>Examples</td>
<td>206</td>
</tr>
<tr>
<td>Errors</td>
<td>207</td>
</tr>
<tr>
<td>Changelog</td>
<td>203</td>
</tr>
<tr>
<td>42. Database Module</td>
<td></td>
</tr>
<tr>
<td>Conventions</td>
<td>209</td>
</tr>
<tr>
<td>General Functions</td>
<td>209</td>
</tr>
<tr>
<td>db:system</td>
<td>209</td>
</tr>
<tr>
<td>db:option</td>
<td>209</td>
</tr>
<tr>
<td>db:info</td>
<td>209</td>
</tr>
<tr>
<td>db:property</td>
<td>209</td>
</tr>
<tr>
<td>db:list</td>
<td>210</td>
</tr>
<tr>
<td>db:list-details</td>
<td>210</td>
</tr>
<tr>
<td>db:backups</td>
<td>210</td>
</tr>
<tr>
<td>Read Operations</td>
<td>211</td>
</tr>
<tr>
<td>db:open</td>
<td>211</td>
</tr>
<tr>
<td>db:open-pre</td>
<td>211</td>
</tr>
<tr>
<td>db:open-id</td>
<td>211</td>
</tr>
<tr>
<td>db:node-pre</td>
<td>211</td>
</tr>
<tr>
<td>db:node-id</td>
<td>211</td>
</tr>
<tr>
<td>db:retrieve</td>
<td>211</td>
</tr>
<tr>
<td>db:export</td>
<td>212</td>
</tr>
<tr>
<td>Value Indexes</td>
<td>213</td>
</tr>
<tr>
<td>db:text</td>
<td>213</td>
</tr>
<tr>
<td>db:text-range</td>
<td>213</td>
</tr>
<tr>
<td>db:attribute</td>
<td>213</td>
</tr>
<tr>
<td>db:attribute-range</td>
<td>213</td>
</tr>
<tr>
<td>db:token</td>
<td>214</td>
</tr>
<tr>
<td>Updates</td>
<td>214</td>
</tr>
<tr>
<td>db:create</td>
<td>214</td>
</tr>
<tr>
<td>db:drop</td>
<td>215</td>
</tr>
<tr>
<td>db:add</td>
<td>215</td>
</tr>
<tr>
<td>db:delete</td>
<td>215</td>
</tr>
<tr>
<td>db:copy</td>
<td>216</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>Functions</td>
<td>264</td>
</tr>
<tr>
<td>http:send-request</td>
<td>264</td>
</tr>
<tr>
<td>Examples</td>
<td>264</td>
</tr>
<tr>
<td>Status Only</td>
<td>264</td>
</tr>
<tr>
<td>Google Homepage</td>
<td>265</td>
</tr>
<tr>
<td>POST Request</td>
<td>266</td>
</tr>
<tr>
<td>Errors</td>
<td>267</td>
</tr>
<tr>
<td>Changelog</td>
<td>267</td>
</tr>
<tr>
<td>51. Index Module</td>
<td>268</td>
</tr>
<tr>
<td>Conventions</td>
<td>268</td>
</tr>
<tr>
<td>Functions</td>
<td>268</td>
</tr>
<tr>
<td>index:facets</td>
<td>268</td>
</tr>
<tr>
<td>index:texts</td>
<td>268</td>
</tr>
<tr>
<td>index:attributes</td>
<td>268</td>
</tr>
<tr>
<td>index:tokens</td>
<td>269</td>
</tr>
<tr>
<td>index:element-names</td>
<td>269</td>
</tr>
<tr>
<td>index:attribute-names</td>
<td>269</td>
</tr>
<tr>
<td>Changelog</td>
<td>269</td>
</tr>
<tr>
<td>52. Inspection Module</td>
<td>270</td>
</tr>
<tr>
<td>Conventions</td>
<td>270</td>
</tr>
<tr>
<td>Reflection</td>
<td>270</td>
</tr>
<tr>
<td>inspect:functions</td>
<td>270</td>
</tr>
<tr>
<td>inspect:function-annotations</td>
<td>270</td>
</tr>
<tr>
<td>inspect:static-context</td>
<td>271</td>
</tr>
<tr>
<td>Documentation</td>
<td>271</td>
</tr>
<tr>
<td>inspect:function</td>
<td>271</td>
</tr>
<tr>
<td>inspect:context</td>
<td>272</td>
</tr>
<tr>
<td>inspect:module</td>
<td>272</td>
</tr>
<tr>
<td>inspect:xqdoc</td>
<td>272</td>
</tr>
<tr>
<td>Examples</td>
<td>273</td>
</tr>
<tr>
<td>Errors</td>
<td>274</td>
</tr>
<tr>
<td>Changelog</td>
<td>275</td>
</tr>
<tr>
<td>53. Jobs Module</td>
<td>276</td>
</tr>
<tr>
<td>Conventions</td>
<td>276</td>
</tr>
<tr>
<td>Services</td>
<td>276</td>
</tr>
<tr>
<td>Basic Functions</td>
<td>276</td>
</tr>
<tr>
<td>jobs:current</td>
<td>276</td>
</tr>
<tr>
<td>jobs:list</td>
<td>276</td>
</tr>
<tr>
<td>jobs:list-details</td>
<td>277</td>
</tr>
<tr>
<td>jobs:finished</td>
<td>277</td>
</tr>
<tr>
<td>jobs:services</td>
<td>277</td>
</tr>
<tr>
<td>Execution</td>
<td>277</td>
</tr>
<tr>
<td>jobs:eval</td>
<td>278</td>
</tr>
<tr>
<td>jobs:invoke</td>
<td>279</td>
</tr>
<tr>
<td>jobs:result</td>
<td>279</td>
</tr>
<tr>
<td>jobs:stop</td>
<td>280</td>
</tr>
<tr>
<td>jobs:wait</td>
<td>280</td>
</tr>
<tr>
<td>Errors</td>
<td>280</td>
</tr>
<tr>
<td>Changelog</td>
<td>280</td>
</tr>
<tr>
<td>54. JSON Module</td>
<td>282</td>
</tr>
<tr>
<td>Conventions</td>
<td>282</td>
</tr>
<tr>
<td>Conversion Formats</td>
<td>282</td>
</tr>
<tr>
<td>Options</td>
<td>283</td>
</tr>
<tr>
<td>Functions</td>
<td>284</td>
</tr>
<tr>
<td>json:parse</td>
<td>284</td>
</tr>
<tr>
<td>json:serialize</td>
<td>284</td>
</tr>
<tr>
<td>Examples</td>
<td>284</td>
</tr>
<tr>
<td>BaseX Format</td>
<td>284</td>
</tr>
<tr>
<td>55. Lazy Module</td>
<td>290</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>lazy:cache</td>
<td>290</td>
</tr>
<tr>
<td>lazy:is-lazy</td>
<td>291</td>
</tr>
<tr>
<td>lazy:is-cached</td>
<td>291</td>
</tr>
<tr>
<td>Changelog</td>
<td>291</td>
</tr>
<tr>
<td>56. Map Module</td>
<td>292</td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>map:contains</td>
<td>292</td>
</tr>
<tr>
<td>map:entry</td>
<td>292</td>
</tr>
<tr>
<td>map:find</td>
<td>293</td>
</tr>
<tr>
<td>map:for-each</td>
<td>293</td>
</tr>
<tr>
<td>map:get</td>
<td>293</td>
</tr>
<tr>
<td>map:keys</td>
<td>294</td>
</tr>
<tr>
<td>map:merge</td>
<td>294</td>
</tr>
<tr>
<td>map:put</td>
<td>294</td>
</tr>
<tr>
<td>map:remove</td>
<td>294</td>
</tr>
<tr>
<td>map:size</td>
<td>295</td>
</tr>
<tr>
<td>Changelog</td>
<td>295</td>
</tr>
<tr>
<td>57. Math Module</td>
<td>296</td>
</tr>
<tr>
<td>Conventions</td>
<td></td>
</tr>
<tr>
<td>W3 Functions</td>
<td></td>
</tr>
<tr>
<td>math:pi</td>
<td>296</td>
</tr>
<tr>
<td>math:sqrt</td>
<td>296</td>
</tr>
<tr>
<td>math:sin</td>
<td>296</td>
</tr>
<tr>
<td>math:cos</td>
<td>296</td>
</tr>
<tr>
<td>math:tan</td>
<td>296</td>
</tr>
<tr>
<td>math:asin</td>
<td>296</td>
</tr>
<tr>
<td>math:acos</td>
<td>297</td>
</tr>
<tr>
<td>math:atan</td>
<td>297</td>
</tr>
<tr>
<td>math:atan2</td>
<td>297</td>
</tr>
<tr>
<td>math:pow</td>
<td>297</td>
</tr>
<tr>
<td>math:exp</td>
<td>297</td>
</tr>
<tr>
<td>math:log</td>
<td>297</td>
</tr>
<tr>
<td>math:log10</td>
<td>297</td>
</tr>
<tr>
<td>Additional Functions</td>
<td></td>
</tr>
<tr>
<td>math:e</td>
<td>298</td>
</tr>
<tr>
<td>math:sinh</td>
<td>298</td>
</tr>
<tr>
<td>math:cosh</td>
<td>298</td>
</tr>
<tr>
<td>math:tanh</td>
<td>298</td>
</tr>
<tr>
<td>math:crc32</td>
<td>298</td>
</tr>
<tr>
<td>Changelog</td>
<td>298</td>
</tr>
<tr>
<td>58. Output Module</td>
<td>299</td>
</tr>
<tr>
<td>Conventions</td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>out:cr</td>
<td>299</td>
</tr>
<tr>
<td>out:nl</td>
<td>299</td>
</tr>
<tr>
<td>out:tab</td>
<td>299</td>
</tr>
<tr>
<td>out:format</td>
<td>299</td>
</tr>
<tr>
<td>Errors</td>
<td></td>
</tr>
<tr>
<td>Changelog</td>
<td>299</td>
</tr>
<tr>
<td>59. Process Module</td>
<td>301</td>
</tr>
</tbody>
</table>
request:context-path ................................................................. 313
Connection Functions .............................................................. 313
request:address .............................................................. 313
request:remote-hostname ......................................................... 314
request:remote-address .......................................................... 314
request:remote-port .............................................................. 314
Parameter Functions .............................................................. 314
request:parameter-names ......................................................... 314
request:parameter .............................................................. 314
Header Functions .............................................................. 314
request:header-names ............................................................ 314
request:header .............................................................. 314
Cookie Functions .............................................................. 315
request:cookie-names ............................................................ 315
request:cookie .............................................................. 315
Changelog ........................................................................... 315
64. RESTXQ Module ............................................................. 316
  Conventions ....................................................................... 316
  General Functions .............................................................. 316
    rest:base-uri .............................................................. 316
    rest:uri .............................................................. 316
    rest:wadl .............................................................. 316
    rest:init .............................................................. 316
  Changelog ........................................................................... 316
65. Session Module ............................................................ 318
  Conventions ....................................................................... 318
  Functions ........................................................................... 318
    session:id .............................................................. 318
    session:created .......................................................... 318
    session:accessed ........................................................ 318
    session:names ............................................................ 318
    session:get .............................................................. 319
    session:set .............................................................. 319
    session:delete ............................................................ 319
    session:close .............................................................. 319
  Errors .............................................................................. 319
  Changelog ........................................................................... 319
66. Sessions Module ............................................................ 321
  Conventions ....................................................................... 321
  Functions ........................................................................... 321
    sessions:ids .............................................................. 321
    sessions:created .......................................................... 321
    sessions:accessed ........................................................ 321
    sessions:names ............................................................ 321
    sessions:get .............................................................. 321
    sessions:set .............................................................. 322
    sessions:delete ............................................................ 322
    sessions:close .............................................................. 322
  Errors .............................................................................. 322
  Changelog ........................................................................... 322
67. SQL Module ..................................................................... 323
  Conventions ....................................................................... 323
  Functions ........................................................................... 323
    sql:init .............................................................. 323
    sql:connect .............................................................. 323
    sql:execute .............................................................. 323
    sql:execute-prepared ....................................................... 324
    sql:prepare .............................................................. 324
user:grant ................................................................. 338
user:drop .................................................................. 339
user:alter ............................................................... 339
user:password ........................................................ 339
user:update-info ...................................................... 339
Errors ..................................................................... 340
Changelog .............................................................. 340
72. Validation Module .................................................. 341
   Conventions .......................................................... 341
   DTD Validation ..................................................... 341
       validate:dtd ...................................................... 341
       validate:dtd-info .............................................. 341
       validate:dtd-report .......................................... 342
   XML Schema Validation ......................................... 342
       validate:xsd ..................................................... 342
       validate:xsd-info ............................................. 343
       validate:xsd-report .......................................... 343
   RelaxNG Validation ............................................... 343
       validate:rng ..................................................... 344
       validate:rng-info ............................................. 344
       validate:rng-report .......................................... 344
   Schematron Validation ........................................... 344
   Errors ................................................................. 344
   Changelog .......................................................... 345
73. Web Module ........................................................ 346
   Conventions ........................................................ 346
   Functions ............................................................ 346
       web:content-type .............................................. 346
       web:create-url ............................................... 346
       web:encode-url .............................................. 346
       web:decode-url ................................................ 346
       web:redirect .................................................... 347
       web:response-header ........................................ 347
   Errors ................................................................. 348
   Changelog .......................................................... 348
74. WebSocket Module .................................................. 349
   Conventions ........................................................ 349
   General Functions ................................................ 349
       ws:id ............................................................ 349
       ws:ids ........................................................... 349
       ws:path ........................................................ 349
       ws:close ........................................................ 349
   Sending Data ........................................................ 349
       ws:send ........................................................ 349
       ws:broadcast .................................................. 350
       ws:emit ........................................................ 350
   WebSocket Attributes ............................................. 350
       ws:get .......................................................... 350
       ws:set .......................................................... 350
       ws:delete ........................................................ 350
   Examples ............................................................ 350
       Example 1 ....................................................... 350
       Example 2 ....................................................... 351
   Errors ................................................................. 351
   Changelog .......................................................... 351
75. XQuery Module ....................................................... 352
   Conventions ........................................................ 352
   Dynamic Evaluation .............................................. 352
Preliminaries ................................................................. 379
Examples ................................................................. 379
Request ................................................................. 380
  Constraints ......................................................... 380
  Content Types ................................................... 382
  Parameters ....................................................... 383
  Query Execution ............................................... 384
Response .............................................................. 385
  Custom Response ................................................ 385
  Forwards and Redirects ....................................... 385
  Output ............................................................ 386
Error Handling ....................................................... 387
  XQuery Errors .................................................. 387
  HTTP Errors .................................................... 388
User Authentication .................................................. 388
Functions ............................................................. 388
References ............................................................ 388
Changelog .............................................................. 389
85. Permissions ........................................................ 390
  Preliminaries ..................................................... 390
  Annotations ...................................................... 390
    Permission Strings ......................................... 390
    Checking Permissions ....................................... 391
  Authentication .................................................. 392
  Changelog ....................................................... 393
86. WebSockets ........................................................ 394
  Introduction ...................................................... 394
    Protocol ........................................................ 394
    Preliminaries ................................................ 394
    Configuration ............................................... 394
  Annotations ...................................................... 395
    %ws:connect(path) ........................................... 395
    %ws:message(path, message) ............................... 395
    %ws:error(path, message) ................................ 395
    %ws:close(path) ............................................. 395
    %ws:header-param(name, variable[, default]) ........... 395
  Writing Applications .......................................... 396
Examples ............................................................. 396
  Basic Example .................................................. 396
  Chat Application ............................................... 397
  Changelog ....................................................... 397
87. REST ................................................................. 398
  Usage ............................................................ 398
  URL Architecture ............................................... 398
    Parameters ................................................... 398
  Request .......................................................... 399
    GET Method .................................................... 399
    POST Method .................................................. 399
    PUT Method .................................................... 400
    DELETE Method ............................................... 401
  Assigning Variables .......................................... 401
    GET Method .................................................... 401
    POST Method .................................................. 402
  Response .......................................................... 402
    Content Type ................................................ 402
Usages Examples ..................................................... 403
  Java ............................................................. 403
  Command Line .................................................... 403
Adjusting the Code ......................................................................................................................... 442
Using the BaseX Android Library .............................................................................................. 445
XI. Advanced User's Guide ......................................................................................................... 446
104. Advanced User's Guide ...................................................................................................... 447
105. Configuration .......................................................................................................................... 448
   Configuration Files ..................................................................................................................... 448
   Home Directory ......................................................................................................................... 448
   Database Directory ..................................................................................................................... 448
   Log Files .................................................................................................................................... 449
   Changelog ................................................................................................................................. 449
106. Backups .................................................................................................................................... 450
   GUI Example ............................................................................................................................. 450
   Console Example ....................................................................................................................... 450
107. Catalog Resolver .................................................................................................................... 451
   Overview ..................................................................................................................................... 451
   XML Entity and URI Resolvers ............................................................................................... 451
   Using other Resolvers ............................................................................................................... 452
   More Information ...................................................................................................................... 452
108. Storage Layout ....................................................................................................................... 453
   Data Types .................................................................................................................................. 453
   Database Files .......................................................................................................................... 453
   Meta Data, Name/Path/Doc Indexes: inf ..................................................................................... 453
   Node Table: tbl, tbl1 .................................................................................................................... 454
   Texts: txt, atv ............................................................................................................................. 454
   Value Indexes: txt1, txtr, atvl, atvr ............................................................................................ 454
   Full-Text Fuzzy Index: fttx, ftxy, fttxz ...................................................................................... 454
109. Node Storage .......................................................................................................................... 455
   Node Table .................................................................................................................................. 455
   PRE Value ................................................................................................................................... 455
   ID Value ....................................................................................................................................... 455
   Block Storage ............................................................................................................................ 455
110. User Management .................................................................................................................... 457
   Rules .......................................................................................................................................... 457
   Operations ................................................................................................................................. 457
   Commands .................................................................................................................................. 457
   XQuery ....................................................................................................................................... 458
   Storage ....................................................................................................................................... 458
   Changelog .................................................................................................................................. 458
111. Transaction Management ....................................................................................................... 459
   Introduction ............................................................................................................................... 459
   XQuery Update .......................................................................................................................... 459
   Concurrency Control ................................................................................................................. 459
   XQuery Locks ............................................................................................................................. 460
   Limitations .................................................................................................................................. 461
   File-System Locks ..................................................................................................................... 461
   Update Operations ..................................................................................................................... 461
   Database Locks ......................................................................................................................... 461
   Changelog .................................................................................................................................. 461
112. Logging .................................................................................................................................... 463
   Introduction ............................................................................................................................... 463
   RESTXQ .................................................................................................................................... 463
   Format ....................................................................................................................................... 463
   Changelog .................................................................................................................................. 464
XII. Use Cases .............................................................................................................................. 465
113. Statistics ................................................................................................................................... 466
   Databases ................................................................................................................................... 466
   Sources ....................................................................................................................................... 468
114. Twitter ..................................................................................................................................... 470
Chapter 1. Main Page

Read this entry online in the BaseX Wiki.

BaseX GUI 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 data (XML, JSON, CSV, many others). With RESTXQ, you can develop full web applications. The visual frontend includes an XQuery editor for running your expressions in realtime, and various visualizations to interactively explore data. BaseX is platform-independent and distributed under the free BSD License (find more in Wikipedia).

This documentation is based on BaseX 9.1. Newest and upcoming features are highlighted and can also be searched for.

If you have questions, or if you want to have direct contact to the developer team and users of BaseX, please write to our basex-talk mailing list. Many questions are being discussed at StackOverflow, and planned features are listed in our GitHub repository.

Getting Started

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

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 own project.

Advanced User’s Guide
Information for advanced users can be found in our advanced user’s guide, which contains details on the BaseX storage, the Client/Server architecture, and some querying features.
Part I. Getting Started
Chapter 2. Getting Started

Read this entry online in the BaseX Wiki.

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

Overview

First Steps

• Startup: How to get BaseX running
• Command-Line Options

User Interfaces

• Graphical User Interface (see available Shortcuts)
• Database Server: The client/server architecture
• Standalone Mode: The command-line interface
• Web Application: The HTTP server
• DBA: Browser-based database administration

General Info

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

Integration

• Integrating oXygen
• Integrating IntelliJ IDEA
• Integrating Eclipse

Tutorials and Slides

BaseX: Introduction

• BaseX for Dummies. Written by Paul Swennenhuys: Part I, Part I (files), Part II.
• BaseX Adventures. Written by Neven Jovanović.
• Tutorial. Written by Imed Bouchrika.
• XQuery pour les Humanités Numériques. Written by Farid Djaidja (French).

XML and XQuery

• XML Technologies. Our university course on XML, XPath, XQuery, XSLT, Validation, Databases, etc.
• XQuery Tutorial. From W3 Schools.
• XQuery: A Guided Tour. From the book "XQuery from the Experts".
• XQuery Summer Institute. Exercises and Answers.

**BaseX: Talks, Questions**

• Our Annual User Meetings. Slides and videos.
• Our Mailing List. Join and contribute.
• GitHub Issue Tracker. Please use our mailing list before entering new issues.
• Stack Overflow. Questions on baseX.
Chapter 3. Startup

Read this entry online in the BaseX Wiki.

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

Getting Started

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

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

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

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

Distributions

Core Package

The Core Package is a small JAR file. It contains the database system, the XQuery processor and the graphical user interface. It does not require any additional libraries.

Full Distributions

The ZIP Package and the Windows Installer contain the BaseX core library and extra libraries for web applications and advanced features, Start Scripts, and some additional optional files.

After BaseX has been unzipped or installed, the following directories will be available:

- bin/: Start scripts (Windows, Linux).
- data/: Database directory. See Configuration for more details.
- etc/: Example data: XML sample, catalog and DTD files.
- lib/: Extra libraries (Jetty, Tagsoup, ...).
- lib/custom/: Directory, in which additional JAR files can be placed (such as the Saxon library).
- repo/: Repository (the FunctX library is included as example).
- src/: Directory for your XQuery scripts and other source data.
- webapp/: Web Application directory: RESTXQ web application, REST scripts, DBA.

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 classpath.

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.

Requirements

BaseX is platform-independent and runs on any system that provides an implementation of the Java Runtime Environment (JRE):

• Since Version 9 of BaseX, Java 8 is required.
• Since Version 8, Java 7 is required.
• Older versions are based on Java 6.

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

Concurrent Operations

If you want to perform parallel (concurrent) read and write operations on your databases, you must use the client/server architecture or deploy BaseX as 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, but 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) at the same time. If you update your data, however, you shouldn’t use the GUI or a standalone instance at the same time.

More details on concurrency are found in the Transaction Management article.

Standalone

The Standalone Mode can be used to execute XQuery expressions or run database commands on command line. It can also be used both for scripting and batch processing your XML data. It can be started as follows (get more information on all Startup Options):

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

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

Graphical User Interface

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

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

• Double click on the BaseX.jar file.
• Run one of the basexgui or basexgui.bat scripts.
• Execute the following command: java -cp BaseX.jar org.basex.BaseXGUI
• On Windows: Double click on the BaseX GUI icon.
• For [Maven](https://mvn.apache.org) users: type in `mvn exec:java` in the main directory of the [basex](https://basex.org) project.

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

**Client/Server**

**Server**

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

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

• Run one of the `basexserver` or `basexserver.bat` scripts. Add the `stop` keyword to gracefully shut down the server.

• Execute the following command: `java -cp BaseX.jar org.basex.BaseXServer`. Again, the `stop` keyword will ensure a graceful shutdown.

• On [Windows](https://windows.microsoft.com): Double click on the [BaseX Server](https://basex.org) icon, which will also start the [HTTP Server](https://httpserver.org), or the [BaseX Server (stop)](https://basex.org) icon.

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

**Client**

The [BaseX Client](https://basex.org) interface can be used to send commands and queries to the server instance on command line.

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

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

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

• On [Windows](https://windows.microsoft.com): Double click on the [BaseX Client](https://basex.org) icon.

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

• **Username:** admin

• **Password:** admin

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

We provide additional clients in various [programming languages](https://programming.com).

**HTTP Server**

With the [HTTP Server](https://httpserver.org), BaseX can be used to build [Web Applications](https://webapp.org). It provides access to the [REST](https://rest.org), [RESTXQ](https://restxq.org) and [WebDAV](https://webdav.org) services. An instance of the [Jetty Web Server](https://jetty.org) will be created, which by default listens to the port 8984. Additionally, the BaseX Server will be started, which is accessible on port 1984.

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

• Run one of the `basexhttp` or `basexhttp.bat` scripts. Call the script with the `stop` keyword to gracefully shut down the server.

• On [Windows](https://windows.microsoft.com): Double click on the [BaseX Server](https://basex.org) or [BaseX Server (stop)](https://basex.org) icon.

• BaseX can also be deployed as [web servlet](https://servlet.org).
In the first two cases, the command-line output will look similar to the following lines (the JSP warning message can be ignored):

```
[main] INFO org.eclipse.jetty.server.AbstractConnector - Started
  SelectChannelConnector@0.0.0.0:8984
HTTP Server was started (port: 8984).
```

You can adjust the Jetty logging level by adding the following properties to the start script:

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

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

**Database Administration**

The DBA is a web-based database administration interface. It allows you to create and administrate databases, evaluate queries in realtime, view log files, manage users, etc.

It can be accessed via http://localhost:8984/dba/.

**Changelog**

Version 8.0

- Update: Switched to Java 7

Version 7.0

- Updated: BaseXJAXRX has been replaced with BaseXHTTP
Chapter 4. Command-Line Options

Read this entry online in the BaseX Wiki.

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

Options can be specified multiple times. 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**

Launch the console mode

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

Available command-line flags can be listed with \(-h\):

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

The meaning of all flags is listed in the following table. If an equivalent database option exists (which can be specified via the \(\text{SET}\) command), it is listed as well. For the examples to work escaping some characters might be necessary, 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>• &quot;doc('X')//head&quot;</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; otherwise, it will be evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td>* query.xq</td>
</tr>
<tr>
<td></td>
<td>• Otherwise, the input string itself is evaluated as XQuery expression.</td>
<td></td>
<td></td>
<td>* commands.bxs</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-b&lt;vars&gt;</code></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 (<code>=</code>). 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></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-c&lt;br&gt;</code></td>
<td>Executes Commands:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Toggles the debugging mode. Debugging information is output to <code>standard error</code>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-i&lt;file&gt;</code></td>
<td>Opens an XML file or database specified by the argument. The opened input can then be processed by a command or XQuery expression.</td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-o&lt;file&gt;</code></td>
<td>All command and query output is written to the specified file.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-q&lt;expr&gt;</code></td>
<td>Executes the specified string as XQuery expression.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-r&lt;num&gt;</code></td>
<td>Specifies how often a specified query will be evaluated.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-R</code></td>
<td>Specifies if a query will be executed or parsed only.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-s&lt;params&gt;</code></td>
<td>Specifies parameters for serializing XQuery results; see <code>Serialization</code> for more details. This flag may be specified multiple times. Key and values are separated by the equality sign (<code>=</code>).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-t</code></td>
<td>Runs all <code>Unit tests</code> in the specified file or directory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-u</code></td>
<td>Propagates updates on input files back to disk.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Prints process and timing information to the <code>standard output</code>.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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>-V</td>
<td>Prints detailed query information to the standard output, including details on the compilation and profiling steps.</td>
<td>QUERYINFO</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>-w</td>
<td>Specifies if whitespaces in XML text nodes should be chopped (which is the default) or preserved.</td>
<td>CHOP</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>-x</td>
<td>This flags turn on the output of the query execution plan, formatted in XML.</td>
<td>XMLPLAN</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>-X</td>
<td>Generates the query plan before or after query compilation. -x needs to be activated to make the plan visible.</td>
<td>COMPPLAN</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>-z</td>
<td>Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.</td>
<td>SERIALIZE</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

**Server**

Launch the server

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

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

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

The flags have the following meaning (equivalent database options are shown in the table as well). For the examples to work escaping some characters might be necessary, 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;cmd&gt;</td>
<td>Executes Commands:</td>
<td>DEBUG</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Several commands in the input can be separated by semicolons (;).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• If the specified input is a valid URL 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>Turns on the debugging mode. Debugging information is output to standard error.</td>
<td></td>
<td></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;num&gt;</td>
<td>Specifies the port on which the server will be addressable.</td>
<td>SERVERPORT</td>
<td>1984</td>
<td>-p9999</td>
</tr>
</tbody>
</table>
Command-Line Options

- **S** Starts the server as service (i.e., in the background). Use YAJSW, or start BaseX as an ordinary background process, to get more options.

- **z** Does not generate any log files.

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

**Client**

Launch the console mode communicating with the server

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

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

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

```
$ basexclient \-h
BaseX [Client]
Usage: basexclient \[-bclnpqrsuvvwxxz\] \[input\]
\[input\] Execute input file or expression
\-b<par> Bind external query variables
\-c<par> Execute commands from file or string
\-d Specify user name
\-o<par> Write output to file
\-p<par> Set server port
\-P<par> Specify user password
\-q<expr> Execute XQuery expression
\-r<num> Set number of query executions
\-R Turn query execution on/off
\-s<par> Set serialization parameter(s)
\-U<par> Specify user name
\-v/V Show (all) process info
\-w Preserve whitespaces from input files
\-x Show query execution plan
\-X Show query plan before/after compilation
\-z Skip output of results
```

The examples to work escaping some characters might be necessary, 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><em>doc('X')/\head</em>' query.xq commands.bxs</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Example</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| `-b<pars>`  | Binds external variables to XQuery expressions. This flag may be specified  | `-b$v=example 
"declare variable $v external; $v" 
- b(URL)ln=value"declare namespace ns='URL'; declare variable $ns:ln external; $ns:ln"` |
| `-c<input>` | Executes Commands:                                                          | `-c"list;info" 
- ccommands.txt 
- c"<info/>"`                                                                   |
| `-d`        | Toggles the debugging mode. Debugging information is output to standard    | `false`                                                                |
| `-i<input>` | Opens an XML file or database specified by the argument. The opened input   | `-iitems.xml 
"//item"`                                                |
| `-I<input>` | Assigns an input string as item of type `xs:untypedAtomic` to the query    | `-I "Hello Universe" - q 
"."`                                       |
| `-n<name>`  | Specifies the host name on which the server is running.                     | `localhost`                                                            |
| `-o<file>`  | All command and query output is written to the specified file.              | `nsserver.basex.org`                                                   |
| `-p<num>`   | Specifies the port on which the server is running.                          | `1984`                                                                 |
| `-P<pass>`  | Specifies the user password. If this flag is omitted, the password will    | `-Uadmin -Padmin`                                                      |
| `-q<expr>`  | Executes the specified string as XQuery expression.                         | `-q"doc('input')//head"`                                               |
| `-r<num>`   | Specifies how often a specified query will be evaluated.                   | `1`                                                                    |
| `-R`        | Specifies if a query will be executed or parsed only.                      | `true`                                                                 |

- **BINDINGS**

- Otherwise, the input string itself is evaluated as XQuery expression.

Note: The examples shown are part of the documentation and are for illustrative purposes only.
Command-Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>-s&lt;pars&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 (=).</td>
<td>SERIALIZER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- smethod=text</td>
</tr>
<tr>
<td>-U&lt;name&gt;</td>
<td>Specifies the user name. If this flag is omitted, the user name will be requested on command line.</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-Uadmin</td>
</tr>
<tr>
<td>-v</td>
<td>Prints process and timing information to the standard output.</td>
<td>false</td>
</tr>
<tr>
<td>-V</td>
<td>Prints detailed query information to the standard output, including details on the compilation and profiling steps.</td>
<td>QUERYINFO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>-w</td>
<td>Specifies if whitespaces in XML text nodes should be chopped (which is the default) or preserved.</td>
<td>CHOP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>chop</td>
</tr>
<tr>
<td>-x</td>
<td>This flags turn on the output of the query execution plan, formatted in XML.</td>
<td>XMLPLAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
</tr>
<tr>
<td>-X</td>
<td>Generates the query plan before or after query compilation. –x needs to be activated to make the plan visible.</td>
<td>COMPPLAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after</td>
</tr>
<tr>
<td>-z</td>
<td>Turns the serialization of XQuery results on/off. This flag is useful if the query is profiled or analyzed.</td>
<td>SERIALIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>true</td>
</tr>
</tbody>
</table>

HTTP Server

Launch the HTTP server

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

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

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

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

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
<th>Option</th>
<th>Default</th>
<th>Examples</th>
</tr>
</thead>
</table>

15
### Command-Line Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stop</code></td>
<td>Stops a local HTTP server and quits. The database server will be stopped as well, unless <code>-l</code> is specified.</td>
</tr>
</tbody>
</table>
| `-c<cmd>` | Executes **Commands**:  
- Several commands in the input can be separated by semicolons (`;`).  
- If the specified input is a valid URL or file reference, this file will be evaluated as **Command Script**. |
| `-d` | Turns on the debugging mode. Debugging information is output to **standard error**. |
| `-h<num>` | Specifies the port on which the HTTP server will be addressable. |
| `-l` | Starts the server in **local mode**, and executes all commands in the embedded database context. |
| `-n<name>` | Specifies the host name on which the server is running. |
| `-p<num>` | Specifies the port on which the database server will be addressable. |
| `-s<num>` | Specifies the port that will be used to stop the HTTP server. |
| `-S` | Starts the server as service (i.e., in the background). Use **YAJSW**, or start BaseX as an ordinary background process, to get more options. |
| `-U<name>` | Specifies a user name, which will be used by the HTTP services for opening a new session. |
| `-z` | Does not generate any **log files**. |

### GUI

Launch the GUI

`$ basexgui [file]`

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

### Changelog

**Version 9.0**
- Added: BaseXHTTP, command-line option `-c`.
- 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 \(-\text{I}\).

Version 8.0

• Removed: Command-line option \(-\text{L}\) (results will now be automatically separated by newlines).

Version 7.9

• Added: Runs tests in file or directory with \(-\text{t}\).
• Removed: interactive server mode.

Version 7.8

• Added: Specify if a query will be executed or parsed only with \(-\text{R}\).

Version 7.7

• Added: Bind host to the BaseX Server with \(-\text{n}\).

Version 7.5

• Added: detection of Command Scripts.
• Removed: HTTP server flags \(-\text{R}\), \(-\text{W}\), and \(-\text{X}\).

Version 7.3

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

Version 7.2

• Added: RESTXQ Service

Version 7.1.1

• Added: Options \(-\text{c}\) and \(-\text{L}\) in standalone and client mode.

Version 7.1

• Updated: Multiple query files and \(-\text{c}/-\text{i}/-\text{q}\) flags can be specified.
Chapter 5. Start Scripts

Read this entry online in the BaseX Wiki.

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

• We recommend you to manually add the `bin` directory of your BaseX directory to the PATH variable of your environment.

• 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.

• The Windows installer automatically adds the project's `bin` directory to your path environment.

• If you work with Maven, you can directly run the scripts in the `basex-core/etc` and `basex-api/etc` sub-directories of the project.

If BaseX terminates with an Out of Memory or Java heap space error, you can assign more RAM via the `-Xmx` flag (see below). The conservative value that was chosen in our distributions ensures that BaseX will also run on 32 bit JVMs.

Main Package

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

Windows: basex.bat

```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
  SRC="$(readlink "$FILE")"
  FILE="$( cd -P "$SRC" )" &&
    FILE="$( cd -P "$SRC" )" && pwd )/
done
MAIN="$( cd -P "$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
java -cp "$CP" $BASEX_JVM org.basex.BaseX "$@"

HTTP Server

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

Windows: basexhttp.bat

@echo off
setLocal EnableDelayedExpansion

REM Path to 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

#!/usr/bin/env bash

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

# 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" $BASEX_JVM org.basex.BaseXHTTP "$@"

Changelog

Version 7.5
• Updated: Static dependencies removed from Windows batch scripts.

Version 7.2
• Updated: The BaseXHTTP start class moved from org.basex.api to org.basex.

Version 7.0
• Updated: The basexjaxrx scripts have been replaced with the basexhttp scripts.
Part II. User Interfaces
Chapter 6. Graphical User Interface

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section.

Startup

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

Select Database → New 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 our full releases (in the etc directory) and can also be downloaded (1.3 MB). If you type nothing in the input field, an empty database will be created. Next, choose the OK button, and the database will be created.

If no XML document is available, the Text Editor can be used to create and edit XML documents. After saving the document to disk, it can be supplied as input for the creation of the database.

Text Editor

The built-in text editor is a powerful tool for writing XQuery scripts and modules, and for developing full RESTXQ applications. It can also be used to edit XML documents, Command Scripts, and any other text files:

- XQuery and XML files will be parsed with each key click, and errors will be highlighted.
- XQuery and command scripts can be executed by clicking on the green triangle.
- Syntax highlighting is available for XQuery, XML, JSON and JavaScript files.

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

Introduced with Version 9.1: If you right-click on an XML document in the Project View, the selected file will be parsed and bound to the context item:
Graphical User Interface

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.

Input Bar

On top of the main window, you can find an input bar. The attached dropdown menu provides three modes: 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>~Cing</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: enter them in the input bar, and press ENTER or click on the START button:

<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/policy &gt; 1000000</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>
Graphical User Interface

Command

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

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

Realtime Options

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

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

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

Visualizations

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

**Text View Result**

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

**Map View Map**

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

**Tree View Tree**

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

**Folder View Folder**

Displays XML data as a file system tree.
Graphical User Interface

Graphical User Interface

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

Scatterplot View

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

The Table View

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

Explorer View

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

Info View

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

Look and Feels

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

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

Changelog

Version 9.1

- Added: Project View, set XML document as context.
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 JTatto library
Chapter 7. 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.

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>Format code (experimental)</td>
<td>Ctrl Shift F</td>
<td># Shift F</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>Version 9.1: 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>
</tbody>
</table>

Finding

<table>
<thead>
<tr>
<th>Description</th>
<th>Win/Linux</th>
<th>Mac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find files (or currently highlighted text)</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>
<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>F3Ctrl G</td>
<td># F3# G</td>
</tr>
<tr>
<td>Find previous instance of text</td>
<td>Shift F3Ctrl Shift G</td>
<td># Shift F3# Shift G</td>
</tr>
</tbody>
</table>
# Shortcuts

## 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/Ctrl Delete</td>
<td># X</td>
</tr>
<tr>
<td>Copy selection to clipboard</td>
<td>Ctrl C/Ctrl Insert</td>
<td># C</td>
</tr>
<tr>
<td>Paste from clipboard</td>
<td>Ctrl V/Shift Insert</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>F6</td>
<td># F6</td>
</tr>
<tr>
<td>Focus editor</td>
<td>F12</td>
<td># F12</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>
<tr>
<td>Reset font size</td>
<td>Ctrl 0</td>
<td># 0</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>
<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, Åuml will be translated to ä).
Changelog

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 8. Database Server

Read this entry online in the BaseX Wiki.

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

Startup

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

Working on Command-Line

The BaseX command-line client provides similar features to the Standalone Mode. 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 transfered 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 -Uadmin -Padmin -c "CREATE DB input <example/>; XQUERY /"
```

Database 'input' created in 13.85 ms.
<example/>
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.
Chapter 9. Standalone Mode

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section. BaseX offers a standalone console mode from which all database commands can be executed. The article on the Database Server provides numerous examples for running commands in the console mode (note that the GUI does not interact with the client/server architecture).

Startup

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

Create a Database

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

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

Where is the database stored?

By default, databases are stored in the basex/data directory, which is located in your home folder. Depending on your Configuration, the location of your home folder varies. For example, on a Mac it's /Users/John, if your name is John.

Execute a Query

The XQUERY command lets you run a query.

• For example, this query returns all country nodes in the currently opened database.
  > XQUERY //country

• You can also run queries in files:
  > RUN /Users/John/query.xq

Database Commands

• The following command lists all databases than 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:open function:
Standalone Mode

> XQUERY db:open("factbook")//country

• To close the current database, please type:

> CLOSE

• Use the DROP DB command to delete a database:

> 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:

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

• Deleting a document from a database is easy:

> DELETE factbook.xml

Make sure that the database, which contains the addressed document, is currently opened.

**Backup and Restore**

• To backup your database, type:

> CREATE BACKUP factbook

• To restore your database, type:

> RESTORE factbook

The backup file is stored in the database directory. The contains the name of the database and a timestamp: [db-name]-[timestamp].zip. If a database is restored, and if several backups exist, the backup with the newest timestamp is taken.
Chapter 10. Web Application

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. The following services are provided:

<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>

Deployment

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

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

Servlet Container

In order to deploy BaseX HTTP Services in a servlet container, you 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:8984/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`.

Configuration

Initial database options can be specified in the `web.xml` file. They need to be defined as context parameters and prefixed with `org.basex..`. The most important options for the web application context are as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>

33
USER | admin
---|---
If a user is specified, no credentials must be passed on by the client.

HTTPLOCAL | false
---|---
Operation mode. By default, a database server instance will be started, as soon as the first HTTP service is called. The database server can be disabled by setting this flag to true.

RESTXQPATH | .
---|---
Relative or absolute directory referencing the RESTXQ modules. By default, the option points to the standard web application directory.

RESTPATH | .
---|---
Relative or absolute directory referencing queries and command-scripts that can be invoked via the run operation of REST. By default, the option points to the standard web application directory.

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

Local file paths in options may be absolute or relative. If a relative path is specified, its root will be the servlet (webapp) path:

```xml
<context-param>
  <param-name>org.basex.dbpath</param-name>
  <!-- will be rewritten to ../../webapp/WEB-INF/data -->
  <param-value>WEB-INF/data</param-value>
</context-param>
<context-param>
  <param-name>org.basex.repopath</param-name>
  <!-- will be kept as is -->
  <param-value>f:/basex/repository</param-value>
</context-param>
```

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**

Different credentials can be assigned to a service by specifying local init parameters. In the following example, an alternative 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 server setting).

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`. An example:

```
http://admin:admin@localhost:8984/
```

Users are specified in a `users.xml` file, which is stored in the database directory (see User Management for more information).
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` sub-directory. This will start a Jetty instance in which the servlets will be deployed.

Configuration

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.

Changelog

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 11. DBA

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section.

The full distributions of BaseX are equipped with a powerful browser-based database administration interface, the **DBA**. It allows you to create and administrate databases, evaluate queries in realtime, view log files, 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 help and motivate you developing your own RESTXQ web applications.
- The XQuery DBA code consumes only around 100 KB. It uses simple backward-compatible Javascript code. The interface is functional, but limited in terms of flashiness and interactivity.
- We tried to make the DBA features as self-explanatory as possible. All functionalities are also available via **Commands**, **XQuery Modules** or the Java GUI.
- The `dba` sub-directory can simply be copied and moved to any other place. All URL paths point to the same directory; it should be straightforward to adjust the RESTXQ path.

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

- you have changed the password of your BaseX admin user, and
- 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`

**First Steps**

On the welcome page, you need to authenticate yourself by entering the name and password of an admin user. The default user is `admin/admin`; after the first login, the password should be changed via the Users panel.

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

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

**Editor**

In the editor panel, you can execute XQuery expressions. If evaluation takes too long, or if it consumes too much memory, it will be interrupted. You need to choose if your query is updating. 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 choosen in the dropdown menu). The realtime mode was removed.
Changelog

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', realtime mode removed.

Introduced with Version 8.0.
Part III. General Info
Chapter 12. Databases

Read this entry online in the BaseX Wiki.

This page is part of the Getting Started Section.

In BaseX, a database is a pretty light-weight concept. 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. Resources can either be XML documents or raw files (binaries). Some information on binary data can be found on an extra page.

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

Create Databases

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

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

The name of a database is restricted to a restricted set of characters (see Valid Names). Various parsers can be chosen to control the import process, or to convert different formats to XML.

**Note**: A main-memory database will be created if the MAINMEM option is enabled (see below for more).

Access Resources

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

**XML Documents**

Various XQuery functions exist to access XML documents in databases:

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db:open</td>
<td><code>db:open(&quot;db&quot;, &quot;path/to/docs&quot;)</code></td>
<td>Returns all documents that are found in the database <code>db</code> at the (optional) path <code>path/to/docs</code>.</td>
</tr>
<tr>
<td>fn:collection</td>
<td><code>collection(&quot;db/path/to/docs&quot;)</code></td>
<td>Returns all documents at the location <code>path/to/docs</code> in the database <code>db</code>. If no path is specified after the database, all documents in the database will be returned. If no argument is specified, all documents of the database will be returned that has been opened in the global context.</td>
</tr>
<tr>
<td>fn:doc</td>
<td><code>doc(&quot;db/path/to/doc.xml&quot;)</code></td>
<td>Returns the document at the location <code>path/to/docs</code> in the database <code>db</code>. 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:
for $i$ in 1 to 100
return db:open('books' || $i)//book/title

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

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db:name</td>
<td>db:name($node)</td>
<td>Returns the name of the database in which the specified $node is stored.</td>
</tr>
<tr>
<td>db:path</td>
<td>db:path($node)</td>
<td>Returns the path of the database document in which the specified $node is stored.</td>
</tr>
</tbody>
</table>

The fn:document-uri and fn:base-uri functions return URIs that can also be reused as arguments for the fn:doc and fn:collection functions. As a result, the following example query always returns true:

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

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

- doc("http://web.de") : retrieves the addressed URI and returns it as a main-memory document node.
- doc("myfile.xml") : retrieves the given file from the file system and returns it as a main-memory document node. Note that updates to main-memory nodes are not automatically written back to disk unless the WRITEBACK option is set.
- collection("/path/to/docs") : returns a main-memory collection with all XML documents found at the addressed file path.

Raw Files

The RETRIEVE command and the db:retrieve function can be used to return files in their native byte representation.

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

string(db:retrieve('multimedia', 'sample.avi'))

HTTP Services

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

Update Resources

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

- XML documents can be added with the ADD command.
- Raw files are added with STORE.
• Existing resources can be replaced with the REPLACE command.
• Resources can be deleted via DELETE.

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

The ADDCACHE option will first cache the input before adding it to the database. This is helpful when the input documents to be added are expected to eat up too much main memory.

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

```
CREATE DB example
SET AUTOFLUSH false
ADD example.xml
SET ADDCACHE true
ADD /path/to/xml/documents
STORE TO images/123.jpg
FLUSH
DELETE /
```

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

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

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

### Export Data

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

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

### Main-Memory Database Instances

• In the standalone context, a main-memory database can be created (using CREATE DB), which can then be accessed by subsequent commands.

• If a BaseX server instance is started, and if a database is created in its context (using CREATE DB), other BaseX client instances can access (and update) this database (using OPEN, db:open, etc.) as long as no other database is opened/created by the server.

• You can force an ordinary database to being copied to memory by using `db:open('some-db') update {}`

Note: If you address a URI with fn:doc or fn:collection for which no database exists, the resulting internal representation is identical to those of main-memory database instances (no matter which value is set for MAINMEM).

### Changelog

Version 8.4
• Updated: **Raw Files**: 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 13. Binary Data

Read this entry online in the BaseX Wiki.

This page is linked from the Database page.

The BaseX store also provides support for raw files (binary data). A database may contain both XML documents and raw files. XML and binary data is handled in a uniform way: a unique database path serves as key, and the contents can be retrieved via database commands, XQuery, or the various APIs.

Storage

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

- **Good Performance**: the file system generally performs very well when it comes to the retrieval and update of binary files.

- **Key/Value Stores**: we do not want to compete with existing key/value database solutions. Again, this is not what we are after.

- **Our Focus**: our main focus is the efficient storage of hierarchical data structures and file formats such as XML or (more and more) JSON. The efficient storage of arbitrary binary resources would introduce many new challenges that would distract us from more pressing tasks.

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

- **Performance Limits**: most file system are not capable of handling thousands or millions of binary resources in a single directory in an efficient way. The same problem happens if you have a large number of XML documents that need to imported in or exported from a BaseX database. The general solution to avoid this bottleneck is to distribute the relevant binaries in additional sub-directories.

- **Keys**: if you want to use arbitrary keys for XML and binary resources, which are not supported by the underlying file system, you may either add an XML document in your database that contains all key/path mappings.

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

Usage

More information on how to store, retrieve, update and export binary data is found in the general Database documentation.
Chapter 14. Parsers

Read this entry online in the BaseX Wiki.

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

XML Parsers

BaseX provides two 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:

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

XQuery

The db:add and db:replace functions can also be used to add new XML documents to the database. The following example query uses the internal XML parser and adds all files to the database DB that are found in the directory 2Bimported:

```
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 this 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,ignorable=true
SET CREATEFILTER *.html
```

**XQuery**

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

Documents can also be converted by specifying the parser and additional options as function arguments:
**JSON Parser**

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

**GUI**

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

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

**Command Line**

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

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

**XQuery**

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

**CSV Parser**

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

**GUI**

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

- **Encoding**: Choose the appropriate encoding of the CSV file.
- **Separator**: Choose the column separator of the CSV file. Possible: comma, semicolon, tab or space or an arbitrary character.
- **Header**: Activate this option if the incoming CSV files have a header line.

**Command Line**

Turn on the CSV Parser before parsing documents, and set some optional, parser-specific options and a file filter. Unicode code points can be specified as separators; 32 is the code point for spaces:

```plaintext
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:

```xquery
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:

```bash
SET PARSER text
SET TEXTPARSER lines=yes
SET CREATEFILTER *
```

**XQuery**

Similar to the other formats, the text parser can also be specified via XQuery:

```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 15. Commands

Read this entry online in the BaseX Wiki.

This article is part of the Getting Started Section. It lists all database commands supported by BaseX.

Commands can 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:

```
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:

```
<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.
Commands

- *AB addresses all names ending with the characters AB.
- X*, Y*, Z* addresses all names starting with the characters X, Y, or Z.

Valid Names

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

- *,?*: glob syntax
- ;: Separator for multiple database commands on the command line
- /: Directory path separators
- :*: 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

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

Examples

- CREATE DB input creates an empty database input.
- CREATE DB xmark http://files.basex.org/xml/xmark.xml creates the database xmark, containing a single initial document called xmark.xml.
- CREATE DATABASE coll /path/to/input creates the database coll with all documents found in the input directory.
- SET INTPARSE false; CREATE DB input.input.xml creates a database input with input.xml as initial document, which will be parsed with Java's default XML parser.
- <create-db name='simple'><hello>Universe</hello></create-db> creates a database named simple with an initial document <hello>Universe</hello>.

OPEN

Syntax OPEN [name] ([path])
Commands

XML Syntax `<open name='...' (path='...')/>

Permission  READ

Summary  Opens the database specified by name. The documents to be opened can be specified by the [path] argument.

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

CHECK

Syntax  CHECK [input]

XML Syntax `<check input='...'/>

Permission  READ/CREATE

Summary  This convenience command combines OPEN and CREATE DB: if a database with the name input exists, it is opened. Otherwise, a new database is created; if the specified input points to an existing resource, it is stored as initial content.

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

CLOSE

Syntax  CLOSE

XML Syntax `<close/>

Permission  READ

Summary  Closes the currently opened database.

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

EXPORT

Syntax  EXPORT [path]

XML Syntax `<export path='...'/>

Permission  CREATE

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

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

CREATE INDEX

Updated with Version 8.4: Token index added.

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]
Commands

**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.

**Administration**

**ALTER DB**

**Syntax** `ALTER DB [name] [newname]`

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

**Permission** CREATE

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

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

**Examples**

- `ALTER DB db tempdb` renames the database `db` into `tempdb`.

**DROP DB**

**Syntax** `DROP DB [name]`

**XML Syntax** `<drop-db name='...'/>`

**Permission** CREATE

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

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

**CREATE BACKUP**

**Syntax** `CREATE BACKUP [name]`

**XML Syntax** `<create-backup name='...'/>`

**Permission** CREATE

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

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

**Examples**

- `BACKUP db` creates a zip archive of the database `db` (e.g. `db-2014-04-01-12-27-28.zip`) in the database directory.

**RESTORE**

**Syntax** `RESTORE [name]`

**XML Syntax** `<restore name='...'/>`

**Permission** CREATE

**Summary** Restores a database with the specified `name`. The name may include the timestamp of the backup file.
<table>
<thead>
<tr>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Errors</strong></td>
</tr>
</tbody>
</table>

**INSPECT**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>INSPECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;inspect/&gt;</code></td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td>READ</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Performs some integrity checks on the opened database and returns a brief summary.</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><code>&lt;drop-backup name='...'/&gt;</code></td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td>CREATE</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Drops all backups of the database with the specified name. The Glob Syntax can be used to address more than one database.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>• DROP BACKUP abc* deletes the backups of all databases starting with the characters abc.</td>
</tr>
</tbody>
</table>

**SHOW BACKUPS**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>SHOW BACKUPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;show-backups/&gt;</code></td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td>CREATE</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows all database backups.</td>
</tr>
</tbody>
</table>

**COPY**

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

**INFO DB**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>INFO DB</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td><code>&lt;info-db/&gt;</code></td>
</tr>
<tr>
<td><strong>Permission</strong></td>
<td>READ</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>Shows general information and meta data 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 |
Commands

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

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

### INFO STORAGE

**Syntax**
INFO STORAGE [start end]

**XML Syntax**
<info-storage (start='...') (end='...')/>

**Permission**
READ

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

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

### Querying

#### LIST

**Syntax**
LIST ([name] ([path]))

**XML Syntax**
<list (name='...' (path='...'))/>

**Permission**
NONE

**Summary**
Lists all available databases. If name is specified, the resources of a database are listed. The output can be further restricted to the resources matching the specified path. If database resources are listed, the size is either the number of nodes (for XML resources) or the number of bytes (for binary resources).

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

#### XQUERY

**Syntax**
XQUERY [query]

**XML Syntax**
<xquery>[query]</xquery>

**Permission**
depends on query

**Summary**
Runs the specified query and prints the result.

**Errors**
The command fails if the specified query is invalid.

**Examples**
- XQUERY 1 to 10 returns the sequence (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).
- SET RUNS 10; XQUERY 1 to 10 returns the results after having run the query 10 times.
- SET XMLPLAN true; XQUERY 1 to 10 returns the result and prints the query plan as XML.

#### RETRIEVE

**Syntax**
RETRIEVE [path]

**XML Syntax**
<retrieve path='...'/>

**Permission**
READ

**Summary**
Retrieves a raw file from the opened database at the specified path.

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>FIND [query]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;find&gt;[query]&lt;/find&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>READ</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if no database is opened.</td>
</tr>
</tbody>
</table>

## TEST

<table>
<thead>
<tr>
<th>Syntax</th>
<th>TEST [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;test path='...'/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>ADMIN</td>
</tr>
<tr>
<td>Summary</td>
<td>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 <code>-t</code> on command line.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if at least one test fails.</td>
</tr>
<tr>
<td>Examples</td>
<td>• TEST project/tests runs all tests in the directory project/tests.</td>
</tr>
</tbody>
</table>

## REPO INSTALL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>REPO INSTALL [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;repo-install path='...'/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>CREATE</td>
</tr>
<tr>
<td>Summary</td>
<td>Installs the package with path path.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• The package to be installed is not a xar file.</td>
</tr>
<tr>
<td></td>
<td>• The package to be installed does not exist or is already installed.</td>
</tr>
<tr>
<td></td>
<td>• The package descriptor is with invalid syntax.</td>
</tr>
<tr>
<td></td>
<td>• The package to be installed depends on a package which is not installed.</td>
</tr>
<tr>
<td></td>
<td>• The package is not supported by the current version of BaseX.</td>
</tr>
<tr>
<td></td>
<td>• A component of the package is already installed as part of another package.</td>
</tr>
</tbody>
</table>

## 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>
</tbody>
</table>
### 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>CREATE</strong></td>
<td>ADD {TO [path]} [input]</td>
<td><code>&lt;add (path='...')&gt;[input]&lt;/add&gt;</code></td>
<td>WRITE</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>
<td>The command fails if the package to be deleted is required by another package.</td>
</tr>
<tr>
<td><strong>DELETE</strong></td>
<td>DELETE [path]</td>
<td><code>&lt;delete path='...'/&gt;</code></td>
<td>WRITE</td>
<td>Deletes all documents from the currently opened database that start with the specified path.</td>
<td>The command fails if no database is opened.</td>
</tr>
<tr>
<td><strong>RENAME</strong></td>
<td>RENAME [path] [newpath]</td>
<td><code>&lt;rename path='...' newpath='...'/&gt;</code></td>
<td>WRITE</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>
<td>The command fails if no database is opened, or if the target path is empty.</td>
</tr>
</tbody>
</table>

### Updates

**ADD**

- **Syntax**: ADD {TO [path]} [input]
- **XML Syntax**: `<add (path='...')>[input]</add>`
- **Permission**: WRITE
- **Summary**: Adds a file, directory or XML string specified by input to the currently opened database at the specified path:
  - input may either be a single XML document, a directory, a remote URL or a plain XML string.
  - A document with the same path may occur than once in a database. If this is unwanted, the REPLACE 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

- **Syntax**: DELETE [path]
- **XML Syntax**: `<delete path='...'/>`
- **Permission**: WRITE
- **Summary**: Deletes all documents from the currently opened database that start with the specified path.
- **Errors**: The command fails if no database is opened.

### RENAME

- **Syntax**: RENAME [path] [newpath]
- **XML Syntax**: `<rename path='...' newpath='...'/>`
- **Permission**: WRITE
- **Summary**: Renames all document paths in the currently opened database that start with the specified path. The command may be used to either rename single documents or directories.
- **Errors**: The command fails if no database is opened, or if the target path is empty.

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

### REPLACE

**Syntax**  
REPLACE [path] [input]

**XML Syntax**  
<replace path='...'>[input]</replace>

**Permission**  
WRITE

**Summary**  
Replaces resources in the currently opened database, addressed by `path`, with the file, directory or XML string specified by `input`, or adds new documents if no resource exists at the specified path.

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

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

### STORE

**Syntax**  
STORE (TO [path]) [input]

**XML Syntax**  
<store (path='...')>[input]</store>

**Permission**  
WRITE

**Summary**  
Stores a raw file specified via `input` in the opened database to the specified `path`:

- The input may either be a file reference, a remote URL, or a plain string.
- If the path denotes a directory, it needs to be suffixed with a slash (`/`).
- An existing resource will be replaced.

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

### OPTIMIZE

**Syntax**  
OPTIMIZE (ALL)

**XML Syntax**  
<optimize/> <optimize-all/>  

**Permission**  
WRITE

**Summary**  
Optimizes the index structures, meta data and statistics of the currently opened database:

- If `ALL` is specified, all database structures are completely reconstructed. The database size will be reduced, and all orphaned data will be deleted.
- 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.
- Database options will be adopted from the original database. Only **AUTOOPTIMIZE** and (if `ALL` is specified) **UPDINDEX** will be adopted from the current options.

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

### FLUSH

**Syntax**  
FLUSH

**XML Syntax**  
<flush/>
Commands

| Permission | WRITE |
| Summary | Explicitly flushes the buffers of the currently opened database to disk. This command is applied if AUTOFLUSH has been set to false. |
| Errors | The command fails if no database is opened. |

Monitoring

SHOW SESSIONS

Syntax

SHOW SESSIONS

XML Syntax

<show-sessions/>

Permission

ADMIN

Summary

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

SHOW USERS

Syntax

SHOW USERS (ON [database])

XML Syntax

<show-users (database='...')/>

Permission

ADMIN

Summary

Shows all users that are 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.

Errors

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

KILL

Syntax

KILL [target]

XML Syntax

<kill target='...'/>

Permission

ADMIN

Summary

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

Errors

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

JOBS LIST

Syntax

JOBS LIST

XML Syntax

<jobs-list/>

Permission

ADMIN

Summary

Returns information on all jobs that are currently queued or executed. See jobs:list-details for more details on the returned table entries.

JOBS RESULT

Syntax

JOBS RESULT [id]

XML Syntax

<jobs-result id='...'/>

Permission

ADMIN

Summary

Returns the cached result of a query with the specified job id:

• Results can only be retrieved once. After retrieval, the cached result will be dropped.
### Commands

**Errors**

- If the original query has raised an error, the cached error will be raised instead.

The command fails if the addressed job is still running or if the result has already been retrieved.

### JOBS STOP

**Syntax**

```plaintext
JOBS STOP [id]
```

**XML Syntax**

```xml
<jobs-stop id='...'/>
```

**Permission**

`ADMIN`

**Summary**

Cancels the execution of a job with the specified id, or drops the cached result of a query. Unknown ids are ignored. All jobs are gracefully stopped; it is up to the process to decide when it is safe to shut down.

### User Management

#### CREATE USER

**Syntax**

```plaintext
CREATE USER [name] ([password])
```

**XML Syntax**

```xml
<create-user name='...'>([password])</create-user>
```

**Permission**

`ADMIN`

**Summary**

Creates a user with the specified name and password. If no password is specified, it is requested via the chosen frontend (GUI or bash).

**Errors**

The command fails if the specified user already exists.

#### ALTER USER

**Syntax**

```plaintext
ALTER USER [name] ([newname])
```

**XML Syntax**

```xml
<alter-user name='...' newname='...'/>
```

**Permission**

`ADMIN`

**Summary**

Renames the user with the specified name to newname.

**Errors**

The command fails if the specified user does not exist, or if the new user already exists.

#### ALTER PASSWORD

**Syntax**

```plaintext
ALTER PASSWORD [name] ([password])
```

**XML Syntax**

```xml
<alter-password name='...'>([password])</alter-password>
```

**Permission**

`ADMIN`

**Summary**

Alters the password of the user with the specified name. If no password is specified, it is requested via the chosen frontend (GUI or bash).

**Errors**

The command fails if the specified user does not exist.

#### DROP USER

**Syntax**

```plaintext
DROP USER [name] (ON [pattern]):
```

**XML Syntax**

```xml
<drop-user name='...'> (pattern='...')</drop-user>
```

**Permission**

`ADMIN`

**Summary**

Drops the user with the specified name. The Glob Syntax can be used to address more than one database or user. If a glob pattern is specified, only the pattern will be removed.

**Errors**

The command fails if admin is specified as user name, or if the specified user does not exist or is currently logged in.
## GRANT

**Syntax**

```
GRANT [NONE|READ|WRITE|CREATE|ADMIN] (ON [pattern]) TO [user]
```

**XML Syntax**

```
<grant name='...' permission='none|read|write|create|admin' (pattern='...')/>
```

**Permission**

ADMIN

**Summary**

Grants the specified permission to the specified user. The Glob Syntax can be used to address more than one user. If a glob pattern is specified, the permission will be applied to all databases that match this pattern.

**Errors**

The command fails if admin is specified as user name or if the specified user does not exist.

**Examples**

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

## PASSWORD

**Syntax**

```
PASSWORD ([password])
```

**XML Syntax**

```
<password>([{password}])</password>
```

**Permission**

NONE

**Summary**

Changes the password of the current user. If no password is specified, it is requested via the chosen frontend (GUI or bash).

## General Commands

**RUN**

**Syntax**

```
RUN [file]
```

**XML Syntax**

```
<run file='...'/>
```

**Permission**

depends on input

**Summary**

Evaluates the contents of file as XQuery expression. If the file ends with the suffix .bxs, the file 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 "create db db1; create db db2"
Commands

- **EXECUTE** "<commands><create-db name='db1'/><create-db name='db2'/></commands>" both commands will create two databases db1 and db2 in a single transaction.

GET

<table>
<thead>
<tr>
<th>Syntax</th>
<th>GET [option]</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;get (option='...')/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>Returns the current value of the Option specified via option. 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>

SET

<table>
<thead>
<tr>
<th>Syntax</th>
<th>SET [option] ([value])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;set option='...')(value)&gt;&lt;/set&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified option is unknown or if the specified value is invalid.</td>
</tr>
</tbody>
</table>

INFO

<table>
<thead>
<tr>
<th>Syntax</th>
<th>INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;info/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>READ</td>
</tr>
<tr>
<td>Summary</td>
<td>Shows global information.</td>
</tr>
</tbody>
</table>

HELP

<table>
<thead>
<tr>
<th>Syntax</th>
<th>HELP ([command])</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;help&gt;([command])&lt;/help&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>If command is specified, information on the specific command is printed; otherwise, all commands are listed.</td>
</tr>
<tr>
<td>Errors</td>
<td>The command fails if the specified command is unknown.</td>
</tr>
</tbody>
</table>

EXIT

<table>
<thead>
<tr>
<th>Syntax</th>
<th>EXIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;exit/&gt;</td>
</tr>
<tr>
<td>Permission</td>
<td>NONE</td>
</tr>
<tr>
<td>Summary</td>
<td>Exits the console mode.</td>
</tr>
</tbody>
</table>

QUIT

<table>
<thead>
<tr>
<th>Syntax</th>
<th>QUIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Syntax</td>
<td>&lt;quit/&gt;</td>
</tr>
</tbody>
</table>
Commands

<table>
<thead>
<tr>
<th>Permission</th>
<th>NONE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Exits the console mode (alias of EXIT).</td>
</tr>
</tbody>
</table>

**Changelog**

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 ATTNNAME 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**).

• Updated: permissions for **REPO DELETE**, **REPO INSTALL** and **REPO LIST**.

Version 7.1

• Updated: **KILL** (killing sessions by specifying IP:port)

Version 7.0

• Added: **FLUSH**, **RETRIEVE**, **STORE**.

• Updated: **ADD**: simplified arguments.
Chapter 16. Options

Read this entry online in the BaseX Wiki.

This page is linked from the Getting Started Section.

The options listed on this page influence the way how database commands are executed and XQuery expressions are evaluated. 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 `GET` 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
  CHOP = false
  ```

- Initial values for global options can also be specified via system properties, which can e.g. be passed on with the `-D` flag on command line, or using `System.setProperty()` before creating a BaseX instance. The specified keys need to be prefixed with `org.basex`. An example:

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

- If using the Mac OS X packaged application then 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.CHOP=false</string>
  </array>
  ```

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

  ```
  <context-param>
    <param-name>org.basex.chop</param-name>
    <param-value>false</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

**Signature** DEBUG [boolean]

**Default** false

**Summary** Sends internal debug info to STDERR. This option can be turned on to get additional information for development and debugging purposes. It can also be triggered on command line via `-d`.

DBPATH

**Signature** DBPATH [path]

**Default** `<code>{home}/data</code>`

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

LOGPATH

**Signature** LOGPATH [path]

**Default** `.logs`

**Summary** Points to the directory in which all log files are stored. Relative paths will be resolved against the `DBPATH` directory.

REPOPATH

**Signature** REPOPATH [path]

**Default** `(home)/repo`

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

LANG

**Signature** LANG [language]

**Default** English

**Summary** Specifies the interface language. Currently, seven languages are available: 'English', 'German', 'French', 'Dutch', 'Italian', 'Japanese', and 'Vietnamese'.

LANGKEY

**Signature** LANGKEY [boolean]

**Default** false

**Summary** Prefixes all texts with the internal language keys. This option is helpful if BaseX is translated into another language, and if you want to see where particular texts are displayed.

FAIRLOCK

**Signature** FAIRLOCK [boolean]
Options

Default false
Summary Defines the locking strategy:

• 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.

• 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

Signature CACHETIMEOUT [seconds]
Default 3600
Summary Specifies how many seconds the results of queries, which have been queued by the asynchronously executed, will be cached in main memory.

Client/Server Architecture

HOST

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

PORT

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

SERVERPORT

Signature SERVERPORT [port]
Default 1984
Summary This is the port the database server will be listening to. This option can also be changed when running the server on command line via -p.

USER

Signature USER [name]
Default empty
Summary Represents a user name, which is used for accessing the server or an HTTP service:

• The default value will be overwritten if a client specifies its own credentials.

• If the default value is empty, login will only be possible if the client specifies credentials.

• The option can also be changed on command line via -U.
**Options**

### PASSWORD

<table>
<thead>
<tr>
<th>Signature</th>
<th>PASSWORD [password]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
</tbody>
</table>
| Summary       | Represents a password, which is used for accessing the server:  
  - The default value will be overwritten if a client specifies its own credentials.  
  - If the default value is empty, login will only be possible if the client specifies credentials.  
  - The option can also be changed on command line via `-P`.  
  - Please note that it is a security risk to specify your password in plain text. |

### AUTHMETHOD

<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>
### Options

**IGNOREHOSTNAME**

<table>
<thead>
<tr>
<th>Signature</th>
<th>IGNOREHOSTNAME [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>If this option is enabled, hostnames of certificates will not be verified. Use IGNORECERT to completely disable certificate verification.</td>
</tr>
</tbody>
</table>

**IGNORECERT**

<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>This option can be turned on to ignore untrusted certificates when connecting to servers. Use IGNOREHOSTNAME to suppress only the hostname verification.</td>
</tr>
</tbody>
</table>

**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>
</tbody>
</table>
| Summary       | Denotes the maximum allowed number of parallel transactions:  
  - If FAIRLOCK is enabled, the number of parallel transactions will never exceed the specified value.  
  - If the option is disabled (which is the default), the limit only applies to transactions that access databases.  
  - 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. |

**LOG**

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

Summary | Turns Logging of server operations and HTTP requests on/off. This option can also be changed when running the server on command line via `-z`.

LOGMSGMAXLEN

<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>

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

<table>
<thead>
<tr>
<th>Signature</th>
<th>WEBPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>(home)/webapp</td>
</tr>
<tr>
<td>Summary</td>
<td>Points to the directory in which all the Web Application contents are stored, including XQuery, Script, RESTXQ and configuration files. This option is ignored if BaseX is deployed as web servlet.</td>
</tr>
</tbody>
</table>

RESTXQPATH

<table>
<thead>
<tr>
<th>Signature</th>
<th>RESTXQPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>Points to the directory which contains the RESTXQ modules of a web application. Relative paths will be resolved against the WEBPATH directory.</td>
</tr>
</tbody>
</table>

PARSERESTXQ

<table>
<thead>
<tr>
<th>Signature</th>
<th>PARSERESTXQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>3</td>
</tr>
</tbody>
</table>
| 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.  
See RESTXQ Preliminaries for more details. |

RESTPATH

<table>
<thead>
<tr>
<th>Signature</th>
<th>RESTPATH [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

HTTPLOCAL

<table>
<thead>
<tr>
<th>Signature</th>
<th>HTTPLOCAL [boolean]</th>
</tr>
</thead>
</table>
### Options

<table>
<thead>
<tr>
<th>Default</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>By default, if BaseX is run as Web Application, a database server instance will be started as well. The server can then be addressed by other BaseX clients in parallel to the HTTP services. If the option is set to true, the database server will be disabled.</td>
</tr>
</tbody>
</table>

### STOPPORT

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

### Create Options

#### General

<table>
<thead>
<tr>
<th>MAINMEM</th>
<th>Signature</th>
<th>Default</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAINMEM [boolean]</td>
<td>false</td>
<td>If this option is turned on, new databases will be created in main memory:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• It is not possible to store binary resources in a main-memory database.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
<td></td>
</tr>
</tbody>
</table>

#### ADDCACHE

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

#### Parsing

#### CREATEFILTER

<table>
<thead>
<tr>
<th>Signature</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATEFILTER [filter]</td>
<td>*.xml</td>
</tr>
<tr>
<td>Options</td>
<td>Summary</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Summary</strong></td>
<td>File filter in the <em>Glob Syntax</em>, which is applied whenever new databases are created, or resources are added to a database.</td>
</tr>
<tr>
<td><strong>ADDArchives</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>ADDArchives [boolean]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>true</code></td>
</tr>
<tr>
<td>Summary</td>
<td>If this option is set to <code>true</code>, files within archives (ZIP, GZIP, TAR, TGZ, DOCX, etc.) are parsed whenever new databases are created or resources are added to a database.</td>
</tr>
<tr>
<td><strong>Archivename</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>Archivename [boolean]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>If this option is set to <code>true</code>, the file name of parsed archives will be included in the document paths.</td>
</tr>
<tr>
<td><strong>SkipCorrupt</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>SkipCorrupt [boolean]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td><strong>AddRaw</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>AddRaw [boolean]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>If this option is activated, and if new resources are added to a database, all files that are not filtered by the <code>CreateFilter</code> option will be added as <em>raw</em> files (i.e., in their binary representation).</td>
</tr>
<tr>
<td><strong>Parser</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>Parser [type]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>XML</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Defines a parser for importing new files to the database. Available parsers are XML, JSON, CSV, TEXT, and HTML. HTML will be parsed as normal XML files if <code>Tagsoup</code> is not found in the classpath.</td>
</tr>
<tr>
<td><strong>CsvParser</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>CsvParser [options]</code></td>
</tr>
<tr>
<td>Default</td>
<td><code>empty</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies the way how CSV data will be parsed. Keys and values are delimited with <code>=</code>, and multiple options are delimited with <code>,</code>. The available options (except for the additional <code>encoding</code> option) are described in the CSV Module.</td>
</tr>
<tr>
<td>Examples</td>
<td><code>encoding=CP1252, header=true</code> parses the input as CP1252 and the first line as header.</td>
</tr>
<tr>
<td><strong>JsonParser</strong></td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td><code>JsonParser [options]</code></td>
</tr>
<tr>
<td>Default</td>
<td></td>
</tr>
</tbody>
</table>
Options

<table>
<thead>
<tr>
<th>Default</th>
<th>empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>format=jsonml,lax=yes interprets the input as JSONML and uses lax parsing.</td>
</tr>
</tbody>
</table>

**HTMLPARSER**

<table>
<thead>
<tr>
<th>Signature</th>
<th>HTMLPARSER [options]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>
| Examples  | • encoding=Shift-JIS,nons=true parses the input as Shift-JIS and suppresses namespaces.  
• lexical=true preserves comments. |

**TEXTPARSER**

<table>
<thead>
<tr>
<th>Signature</th>
<th>TEXTPARSER [options]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>lines=true creates a single element for each line of text.</td>
</tr>
</tbody>
</table>

**XML Parsing**

**CHOP**

<table>
<thead>
<tr>
<th>Signature</th>
<th>CHOP [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
</tbody>
</table>
| Summary   | Many XML documents include whitespaces that have been added to improve readability. This option controls the white-space processing mode of the XML parser:  
• With the default value true, leading and trailing whitespaces from text nodes will be chopped and all empty text nodes will be discarded.  
• The flag should be turned off if a document contains mixed content.  
• The flag can also be turned off on command line via -w.  
• If the xml:space="preserve" attribute is attached to an element, chopping will be turned off for all descendant text nodes.  
In the following example document, the whitespaces in the text nodes of the text element will not be chopped: |

```xml
<xml>
  <title>
    Demonstrating the CHOP flag
  </title>
  <text xml:space="preserve">To <b>be</b>, or not to <b>be</b>, that is the question.</text>
</xml>
```
It is recommendable to additionally assign indent=no to the SERIALIZER option; otherwise the serialized documents will automatically be indented.

**STRIPNNS**

<table>
<thead>
<tr>
<th>Signature</th>
<th>STRIPNNS [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>Strips all namespaces from an XML document and all elements while parsing.</td>
</tr>
</tbody>
</table>

**INTPARSE**

<table>
<thead>
<tr>
<th>Signature</th>
<th>INTPARSE [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
</tbody>
</table>
| 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**

<table>
<thead>
<tr>
<th>Signature</th>
<th>DTD [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</td>
<td>Parses referenced DTDs and resolves XML entities. By default, this option is switched to false, as many DTDs are located externally, which may completely block the process of creating new databases. The CATFILE option can be changed to locally resolve DTDs.</td>
</tr>
</tbody>
</table>

**XINCLUDE**

<table>
<thead>
<tr>
<th>Signature</th>
<th>XINCLUDE [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Summary</td>
<td>Resolves XInclude inclusion tags and merges referenced XML documents. By default, this option is switched to true. This option is only available if the standard Java XML Parser is used (see INTPARSE).</td>
</tr>
</tbody>
</table>

**CATFILE**

<table>
<thead>
<tr>
<th>Signature</th>
<th>CATFILE [path]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
<tr>
<td>Summary</td>
<td>Specifies a catalog file to locally resolve DTDs. See the entry on Catalog Resolvers for more details.</td>
</tr>
</tbody>
</table>

**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]
---|---
**Default** | empty
**Summary** | 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.

### ATTRINCLUDE

**Signature** | ATTRINCLUDE [names]
---|---
**Default** | empty
**Summary** | 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.

### TOKENINCLUDE

**Signature** | TOKENINCLUDE [names]
---|---
**Default** | empty
**Summary** | 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.

### FTINCLUDE

**Signature** | FTINCLUDE [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></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 of strings that are to be indexed by the name, path, value, and full-text index structures. The value of this option will be assigned once to a new database, and cannot be changed after that.</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>
</tbody>
</table>
| **UPDINDEX** | UPDINDEX [boolean] | false       | If turned on, incremental indexing will be enabled:  
  - 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($db, true()).  
  - 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** | AUTOOPTIMIZE [boolean] | false       | 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** | SPLITSIZE [num] | 0           |                                                                                                                                                                                                           |
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:

- The larger the assigned value is, the less splits will take place, and the more main memory will be required.
- 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.

### 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.

#### DIACRITICS

**Signature** DIACRITICS [boolean]

**Default** false

**Summary** If set to true, diacritics will be preserved during full-text indexing. By default, diacritics will be removed. See Full-Text Index for more details.

#### LANGUAGE

**Signature** LANGUAGE [lang]

**Default** en

**Summary** 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.

#### STOPWORDS

**Signature** STOPWORDS [path]

**Default** empty

**Summary** A new full-text index will drop tokens that are listed in the specified stopword list. A stopword list may decrease the size of the full text index. See Full-Text Index for more details.

### Query Options

#### QUERYINFO

**Signature** QUERYINFO [boolean]
Options

<table>
<thead>
<tr>
<th>Default</th>
<th>false</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</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>
</tr>
</tbody>
</table>

XQUERY3

<table>
<thead>
<tr>
<th>Signature</th>
<th>XQUERY3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>true</td>
</tr>
<tr>
<td>Summary</td>
<td>Enables all XQuery 3.0 features supported by BaseX. If this option is set to false, the XQuery parser will only accept expressions of the XQuery 1.0 specification.</td>
</tr>
</tbody>
</table>

MIXUPDATES

<table>
<thead>
<tr>
<th>Signature</th>
<th>MIXUPDATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>Summary</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, this option is set to false, because the XQuery Update Facility does not allow an updating query to return results.</td>
</tr>
</tbody>
</table>

BINDINGS

<table>
<thead>
<tr>
<th>Signature</th>
<th>BINDINGS [vars]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>empty</td>
</tr>
</tbody>
</table>
| Summary   | Contains external variables to be bound to a query. The string must comply with the following rules:  
  • Variable names and values must be separated by equality signs.  
  • Multiple variables must be delimited by commas.  
  • Commas in values must be duplicated.  
  • Variables may optionally be introduced with a leading dollar sign.  
  • If a variable uses a namespace different to the default namespace, it can be specified with the Clark Notation or Expanded QName Notation.  
This option can also be used on command line with the flag -b. |
| Examples  | $a=1,$b=2  binds the values 1 and 2 to the variables $a and $b  
$=1,,2  binds the value 1,2 to the variable $a  
(URI)a=x  binds the value x to the variable $a with the namespace URI.  
In the following Command Script, the value hello world! is bound to the variable $GREETING: |

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

INLINELIMIT

<table>
<thead>
<tr>
<th>Signature</th>
<th>INLINELIMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>100</td>
</tr>
</tbody>
</table>

76
**Options**

<table>
<thead>
<tr>
<th>Summary</th>
<th>This option controls inlining of XQuery functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• The XQuery compiler inlines functions to speed up query evaluation.</td>
</tr>
<tr>
<td></td>
<td>• Inlining will only take place if a function body is not too large (i.e., if it does not contain too many expressions).</td>
</tr>
<tr>
<td></td>
<td>• With this option, this maximum number of expressions can be specified.</td>
</tr>
<tr>
<td></td>
<td>• Function inlining can be turned off by setting the value to 0.</td>
</tr>
<tr>
<td></td>
<td>• The limit can be locally overwritten via the <code>%baseX:inline</code> annotation (follow the link to get more information on function inlining).</td>
</tr>
</tbody>
</table>

**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>

**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 <code>fn:doc</code> and <code>fn:collection</code> functions will first be resolved against a database that has been opened in the global context outside the query (e.g. by the <code>OPEN</code> 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 <code>fn:doc</code> and <code>fn:collection</code>.</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>
<tr>
<td>Summary</td>
<td>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 <code>archive:create</code>, <code>archive:extract-text</code>, <code>archive:update</code>, and <code>zip:text-entry</code>.</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 the BaseX-specific fuzzy match option. See the page on Full-Texts for more information on fuzzy querying.</td>
</tr>
</tbody>
</table>
Options

---

**RUNQUERY**

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

---

**RUNS**

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

---

**ENFORCEINDEX**

**Signature** | ENFORCEINDEX [boolean]
---|---
**Default** | false
**Summary** | Enforces index rewritings in path expressions (see Enforce Rewritings for details).

---

**COPYNODE**

**Signature** | COPYNODE [boolean]
---|---
**Default** | true
**Summary** | When creating new nodes in XQuery via Node Constructors, all enclosed nodes will be copied, and all resulting nodes will get new node identities. This step can be very expensive, and it can be disabled with this option. The option should be used carefully, as it changes the standard behavior of XQuery. It should preferably be used in Pragmas.

---

**Serialization Options**

---

**SERIALIZE**

**Signature** | SERIALIZE [boolean]
---|---
**Default** | true
**Summary** | Results of XQuery expressions will be serialized if this option is turned on. For debugging purposes and performance measurements, this option can be set to false. It can also be turned off on command line via -z.

---

**SERIALIZER**

**Signature** | SERIALIZER [params]
---|---
**Default** | empty
**Summary** | Parameters for serializing query results. The string must comply with the following rules:
- Variable names and values must be separated by equality signs.
- Multiple variables must be delimited by commas.
- Commas in values must be duplicated.

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

**Examples**
- `indent=no` : disables indentation of XML nodes (by default, this is enabled)
- `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**

<table>
<thead>
<tr>
<th>Signature</th>
<th>EXPORTER [params]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>empty</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Contains parameters for exporting all resources of a database. Keys and values are separated by equality signs, multiple parameters are delimited by commas. See <a href="#">Serialization</a> for more details.</td>
</tr>
</tbody>
</table>

**XMLPLAN**

<table>
<thead>
<tr>
<th>Signature</th>
<th>XMLPLAN [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Prints the execution plan of an XQuery expression in its XML representation. This option can also be activated on command line via <code>-x</code>.</td>
</tr>
</tbody>
</table>

**COMPPLAN**

<table>
<thead>
<tr>
<th>Signature</th>
<th>COMPPLAN [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>true</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Generates the query plan, which can be activated via XMLPLAN, before or after query compilation. This option can also be activated on command line via <code>-X</code>.</td>
</tr>
</tbody>
</table>

**DOTPLAN**

<table>
<thead>
<tr>
<th>Signature</th>
<th>DOTPLAN [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Saves the query plan of an XQuery expression as <code>.dot</code> file in the current working directory, using the graph description language. The output file can e.g. be visualized with Graphviz.</td>
</tr>
</tbody>
</table>

**DOTCOMPACT**

<table>
<thead>
<tr>
<th>Signature</th>
<th>DOTCOMPACT [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>false</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Chooses a compact dot representation.</td>
</tr>
</tbody>
</table>

**Other Options**

**AUTOFLUSH**

<table>
<thead>
<tr>
<th>Signature</th>
<th>AUTOFLUSH [boolean]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>true</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Flushes database buffers to disk after each update. If this option is set to <code>false</code>, 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.</td>
</tr>
</tbody>
</table>
Options

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. This option can also be activated on command line via `-u`. Please note that, when turning this option on, your original files will not be backed up.

MAXSTAT

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

Changelog

Version 9.0
• Added: ENFORCEINDEX, COPYNODE, IGNOREHOSTNAME

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

- Updated: PROXYPORT: default set to 0; will be ignored. PROXYHOST, NONPROXYHOSTS: empty strings will be ignored.

Version 7.8.1
- Updated: ADDARCHIVES: parsing of TAR and TGZ files.

Version 7.8
- Added: CSVPARSER, JSONPARSER, TEXTPARSER, HTMLPARSER, INLINELIMIT, TAILCALLS, DEFAULTDB, RUNQUERY
- Updated: WRITEBACK only applies to main-memory document instances.
- Updated: DEBUG option can be changed at runtime by users with admin permissions.
- Updated: default of INTPARSE is now false.
- Removed: HTMLOPT (replaced with HTMLPARSER), PARSEROPT (replaced with parser-specific options), DOTDISPLAY, DOTTY

Version 7.7
- Added: ADDCACHE, CHECKSTRINGS, FTINDEXSPLITSIZE, INDEXSPLITSIZE

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

Version 7.5
- Added: options can now be set via system properties
- Added: a pragma expression can be used to locally change database options
- Added: USER, PASSWORD, LOG, LOGMSGMAXLEN, WEBPATH, RESTXQPATH, HTTPLOCAL, CREATEONLY, STRIPNS
- Removed: HTTPPATH; HTTPPORT: jetty.xml configuration file is used instead
- Removed: global options cannot be changed anymore during the lifetime of a BaseX instance

Version 7.3
- Updated: KEEPALIVE, TIMEOUT: default values changed
- Removed: WILDCARDS; new index supports both fuzzy and wildcard queries
- Removed: SCORING; new scoring model will focus on lengths of text nodes and match options

Version 7.2
- Added: PROXYHOST, PROXYPORT, NONPROXYHOSTS, HTMLOPT
- Updated: TIMEOUT: ignore timeout for admin users

Version 7.1
- Added: ADDRAW, MAXLEN, MAXCATS, UPDINDEX
- Updated: BINDINGS
Version 7.0

- Added: SERVERHOST, KEEPALIVE, AUTOFLUSH, QUERYPATH
Part IV. Integration
Chapter 17. Integrating oXygen

Read this entry online in the BaseX Wiki.

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

- Resources in databases can be opened and modified.
- XPath/XQuery 3.0 expressions can be run by the query processor of BaseX.

Access Database Resources

Preparations

- First, start the BaseX WebDAV service.

Configuration

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

You can then access the database files via the Data Source Explorer: Windows → Show View → Data Source Explorer
Perform Queries

One-Time Setup

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

Configure Data Source
1. Start oXygen and 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) as type.
4. Add the following JAR files (downloaded in Preparations procedure) with the Add Files Button. The versions of the JAR files may differ.
   • lib/xqj-api-1.0.jar
   • lib/xqj2-0.2.0.jar
   • lib/basex-xqj-8.6.jar
   • basex.jar, if you want to use BaseX embedded
5. Under "Driver class", choose the preferred driver class:
   • Embedded: net.xqj.basex.BaseXXQDataSource
   • Client/server: net.xqj.basex.local.BaseXXQDataSource
6. Click OK.

Configure Connection
1. In the Connections section (in the same Preferences dialog from the Configure Data Source procedure), click New (+).
2. Enter "BaseX" as name and select "BaseX XQJ" as data source.
3. If you use the default driver, enter these values in the Connection Details section:
Integrating oXygen

- port: 1984
- serverName: localhost
- user: admin
- password: admin

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.

2. Select the XQuery transformation tree entry, and click + to add a new scenario.
   
   If this entry does not appear in the tree, click + and select XQuery transformation in the dropdown list.

3. Enter a name and an optional XML and XQuery URL (e.g. your query document/file).

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

5. Click OK to complete the scenario configuration.

**Execute Query**

After the one-time setup steps are complete, execute a query using the transformation scenario. Start the transformation by clicking the red Apply associated scenarios button in the Transformation Scenarios window.

The results should immediately occur in the result panel.
Chapter 18. 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.

**Installation**

The following steps apply to all operating systems:

- Install either version of IntelliJ IDEA: the Community or Ultimate edition.

This article focuses on Grzegorz Ligas' XQuery Support plugin.

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

**From the Start Screen**

1. Start IDEA and select Configure → Plugins.
2. In the Plugins window select Browse Repositories.
3. In the Browse Repositories window, search for (or scroll to) XQuery Support.
4. You will be prompted to restart IDEA to load the new plugin.

**From the File Menu**

1. Select Settings.
2. In the Settings window select Plugins.
3. In the Plugins window select Browse Repositories.
4. In the Browse Repositories window, search for (or scroll to) XQuery Support.
5. You will be prompted to restart IDEA to load the new plugin.

**Setting Up**

**File Extensions and XQuery Flavor**

- Start IDEA and navigate to Settings (OS dependent).

- In the Settings window select Other Settings → XQuery and choose which XQuery flavor you would like to use, as well as new file extensions. (see attached screenshot)

**Configuring The Processor**

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

**Standalone**

1. Start IDEA and navigate to Settings (OS dependent).
2. In the settings window select Other Settings → XQuery Data Sources
3. Select BaseX (native embedded).

4. Check the User defined XQJ Driver radio box.

5. Using the +, add the following jars: BaseX.jar, basex-api-8.6.1.jar, basex-xqj-8.6.jar, and xqj2-0.2.0.jar from your BaseX distribution.

Client

This assumes that you already have a database named factbook.

1. Start IDEA and navigate to Settings (OS dependent).

2. In the settings window select Other Settings → XQuery Data Sources.

3. Using the +, select BaseX as your data source.

4. Fill in the appropriate connection details; e.g. default values:
   - Host = localhost
   - Port = 1984
   - Database name = factbook
   - Username = admin
   - Password = admin

5. Select Apply, then OK and your BaseX factbook database is ready to query.
Chapter 19. Integrating Eclipse

Read this entry online in the BaseX Wiki.

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

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

Installation

The following steps apply to all operating systems:

- Install Version 3.7 (Indigo) of Eclipse: http://www.eclipse.org. Please note that more recent versions may work as well, but haven’t been tested so far.

- download your favorite BaseX distribution (JAR, ZIP, EXE): http://basex.org/download/

Windows

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

Linux

- First, install the Dynamic Languages Toolkit (DLTK)Update Site: http://download.eclipse.org/releases/indigo/

- Next, install Marklogic’s XQDT Dropin

Mac OSX

- Install Marklogic’s XQDT Dropin
Setting up

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

Setting up as Standalone

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

Setting up as Client

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

The query execution works as follows:

1. Create a new XQuery Project with File → New → XQuery Project.
3. Edit your XQuery Module and execute it with Run.
4. The results are displayed in the Console window of Eclipse.
Part V. Query Features
Chapter 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. However, the processor is also a flexible general purpose processor, which can access local and remote sources. High conformance with the official specifications is one of our main objectives, as the results of the XQuery Test Suite demonstrate. This section contains information on the query processor and its extensions:

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

XQuery Extensions
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.

Updates
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:open('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:

```xml
<challenge>4731</challenge>
<na>12677</na>
<preferred>314</preferred>
<standard>7778</standard>
```

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

Result:

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

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

XQuery:

let $data := 
<xml>
<person country='USA' name='John'/>
<person country='USA' name='Jack'/>
<person country='Germany' name='Johann'/>
</xml>
for $person in $data/person
  group by $country := $person/@country/string()
return element persons {
  attribute country { $country },
  $person/@name ! element name { data() }
}

Result:

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

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:
  
  ```
  let $f := function($x, $y) { $x + $y } 
  return $f(17, 25)
  
  Result: 42
  ```

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

  ```
  let $f := math:pow#2 
  return $f(5, 2)
  
  Result: 25
  ```

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

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

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

**Simple Map Operator**

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

**Example:**

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

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:

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

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

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

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

Switch

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

Example:

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

**Result:** red green pink

The expression to evaluate can correspond to multiple input values.

Example:

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

Result: red red

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'.

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.

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:

declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
declare option output:omit-xml-declaration "no";
declare option output:method "xhtml";
<html/>

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

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

Context Item

The context item can 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)
```

Result: 5 5

Annotations

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

Example:

```xml
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:fold-left
- fn:fold-right
- fn:for-each
- fn:for-each-pair
- fn:format-date
- fn:format-dateTime
- fn:format-integer
- fn:format-number
- fn:format-time
- fn:function-arity
- fn:functionlookup
- 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:datatype

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:

```xml
declare function local:for-each(
    $seq as item()*,
    $fun as function(item()) as 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 }) |
**Higher-Order Functions**

**Result:** 2 4 6 8 10

- Strings that start with an upper-case letter:

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

**Result:** FooBar BAR

- Inefficient prime number generator:

```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**

`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) { $x mod 2 }),(1, 1, 1, 1),
  function($a, $b) { $a + $b }
)
```

**Result:** 2 1 2 1 2

- Line numbering:

```xquery
```
Higher-Order Functions

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

Result:

1: how
2: are
3: you?

• Checking if a sequence is sorted:

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

Result: true false

XQuery 1.0

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

Folds

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

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

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

```java
public int product(int[] seq) {
    int result = 1;
```
Higher-Order Functions

```java
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**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>fn:fold-left($seq as item()<em>), $seed as item()</em>), $function as function(item()<em>, item()) as item()</em>) as item()*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>The left fold traverses the sequence from the left. The query fn:fold-left(1 to 5, 0, $f) for example would be evaluated as:</td>
</tr>
<tr>
<td></td>
<td>$f($f($f($f($f(0, 1), 2), 3), 4), 5)</td>
</tr>
<tr>
<td>Examples</td>
<td>• Product of a sequence of integers:</td>
</tr>
<tr>
<td></td>
<td>fn:fold-left(1 to 5, 1, $f)</td>
</tr>
<tr>
<td></td>
<td>function($result, $curr) { $result * $curr }</td>
</tr>
<tr>
<td></td>
<td>Result: 120</td>
</tr>
<tr>
<td></td>
<td>• Illustrating the evaluation order:</td>
</tr>
<tr>
<td></td>
<td>fn:fold-left(1 to 5, &quot;$seed&quot;, $seed, concat(&quot;$f('', '?', ' ', '?', ')''))</td>
</tr>
<tr>
<td></td>
<td>Result: $f($f($f($f($f($seed, 1), 2), 3), 4), 5)</td>
</tr>
<tr>
<td></td>
<td>• Building a decimal number from digits:</td>
</tr>
<tr>
<td></td>
<td>let $from-digits := fold-left(?), 0, $n, $d) { 10 * $n + $d }</td>
</tr>
<tr>
<td></td>
<td>return (</td>
</tr>
<tr>
<td></td>
<td>$from-digits(1 to 5), $from-digits((4, 2))</td>
</tr>
<tr>
<td></td>
<td>Result: 12345 42</td>
</tr>
</tbody>
</table>

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

```xquery
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
    )
}
```
Higher-Order Functions

fn:fold-right

Signatures

\[
\text{fn:fold-right($seq$ as item()*, $seed$ as item()*, $function$ as function(item(), item()*) as item()*) as item()*)}
\]

Summary

The right fold \( \text{fn:fold-right($seq$, $seed$, $fun$) \text{ traverses the sequence from the right.}} \) The query \( \text{fn:fold-right(1 to 5, 0, $f$) \text{ for example would be evaluated as:}} \)

\[
\$f(1, \$f(2, \$f(3, \$f(4, \$f(5, 0))))))
\]

Examples

- Product of a sequence of integers:

\[
\text{fn:fold-right(1 to 5, 1, function($curr$, $result$) { $result$ * $curr$ })}
\]

\text{Result: 120}

- Illustrating the evaluation order:

\[
\text{fn:fold-right(1 to 5, '$seed', concat('$f(', ?, ', ', ?, ')'))}
\]

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

- Reversing a sequence of items:

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

\text{Result: 10 9 8 7 6 5 4 3 2 1}

XQuery 1.0

\[
\text{declare function local:fold-right(}$seq$ as item()*, $seed$ as item()*, $function$ as function(item(), item()*) as item()*) as item()*) { if(empty($seq)) then $seed else $function{ fn:head($seq), local:fold-right(tail($seq), $seed, $function) }})
\]

Note that the order of the arguments of $fun$ are inverted compared to that in \text{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:

```xml
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:

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

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

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

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

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

### Atomization

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

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

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

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

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

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

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

### Lookup Operator

The lookup operator provides some syntactic sugar to access values of maps or array members. 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(.) :),
    $map?R          (: 2. returns the value associated with the key 'R'; same as: $map('R') :),
    $map?('G','B')  (: 3. returns the values associated with the key 'G' and 'B' :) ),

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

The lookup operator can also be used without left operand. In this case, the context item will be used as input.

This query returns `Akureyri`:

```xquery
let $maps := ( map { 'name': 'Guðrún', 'city': 'Reykjavík' },
               map { 'name': 'Hildur', 'city': 'Akureyri' } )
return $maps[?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 $v => f(v, j)$ is equivalent to $f(v, j)$:

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

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

(: Returns xfmdpnf :) codepoints-to-string( for $i in string-to-codepoints('welcome') returning $i + 1 )
```

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

### 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:

```
``["This is a 'new' & 'flexible' syntax."]``

XQuery expressions can be embedded via backticks and a curly bracket. The evaluated results will be separated with spaces, and all strings will eventually be concatenated:

```
(: Returns »Count 1 2 3, and I will be there.« :) 
let $c := 1 to 3 
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" }
```
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.

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

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).

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

fn:json-to-xml

Signatures

• fn:json-to-xml($string as xs:string?) as node()?
Converting a JSON string to an XML node representation. The allowed options can be looked up in the specification.

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

(: 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
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()*, $collation as xs:string?) as xs:anyAtomicType*()) as item()*`
- `fn:sort($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.

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

---

**fn:contains-token**

Signatures

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

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

```xquery
contains-token(('a', 'b c', 'd'), 'c') (: yields true :)
```
XQuery 3.1

<xml class='one two' />/contains-token(@class, 'one') (: yields true :)  

**fn:parse-ietf-date**

Signature

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

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

| fn:parse-ietf-date('Wed, 01 Jun 2001 23:45:54 +02:00') | (: yields 2001-06-01T23:45:54+02:00 :) |

**fn:apply**

Signatures

- fn:apply($function as function(*), $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:

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

**fn:random-number-generator**

Signatures

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

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

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

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

Example:

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

The function has been extended to support scientific notation:

\[
\text{format-number}(1984.42, '00.0e0') \quad (: \text{yields } 19.8e2 :) 
\]

fn:tokenize

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

\[
\text{fn:tokenize(" a b c d")} \quad (: \text{yields } "a", "b", "c", "d" :) 
\]

fn:trace

The second argument can now be omitted:

\[
\text{fn:trace(<xml/>, "Node: "/node() \quad (: \text{yields the debugging output } "Node: <xml/"> :) 
\]

\[
\text{fn:trace(<xml/>)/node()} \quad (: \text{returns the debugging output } "<xml/>" :) 
\]

fn:string-join

The type of the first argument is now \( \text{xs:anyAtomicType}\ast \), and all items will be implicitly cast to strings:

\[
\text{fn:string-join(1 to 3)} \quad (: \text{yields the string } "123" :) 
\]

fn:default-language

Returns the default language used for formatting numbers and dates. BaseX always returns \( \text{en} \).

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 \( \text{xs:hexBinary} \) and \( \text{xs:base64Binary} \) can be compared against each other. The following queries all yield \( \text{true} \):

\[
\begin{align*}
\text{xs:hexBinary('') & < xs:hexBinary('bb'),} \\
\text{xs:hexBinary('aa') & < xs:hexBinary('bb'),} \\
\text{max((xs:hexBinary('aa'), xs:hexBinary('bb'))) & = xs:hexBinary('bb')} \\
\end{align*}
\]

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 \( \text{true} \):

\[
\begin{align*}
\text{declare default collation 'http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive';} \\
'\text{HTML}' & = 'html' \\
\end{align*}
\]

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

```
declare function local:x() { () }; i
try { () } catch * { () },
element x { () },
text { () }
```

With XQuery 3.1, the expression can be omitted. The following query is equivalent to the upper one:

```
declare function local:x() { }; i
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

*Introduced with Version 9.1:* ternary if, elvis operator, if without else

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.

```
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:

```
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 $b 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 $b is true'  
)

The behavior of the if expression is equivalent to the util:if function.

Functions

Regular Expressions

*Introduced with Version 9.1:*

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 $\backslash b$, which has not been included in the XQuery grammar for regular expressions:

Example:

```xml
(: 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.

- `csv` allows you to output XML nodes as CSV data (see the CSV Module for more details).

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:

```xml
declare option db:chop 'false';  
doc('doc.xml')
```

XQuery Locks

If XQuery Locks are defined in the query prolog of a module, access to functions of this module locks will be controlled by the central transaction management.

If the following XQuery code is called by two clients in parallel, the queries will be evaluated one after another:

```xml
declare option basex:write-lock 'CONFIGLOCK';  
file:write('config.xml', <config/>)
```

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 100000000 })

This pragma can be helpful when debugging your code.

Introduced with Version 9.1:

In analogy with option declarations and function annotations, XQuery Locks can also set via pragmas:

(# basex:write-lock CONFIGLOCK #) { file:write('config.xml', <config/>) }

Database Pragmas

All local options can be assigned via pragmas. Some examples:

• Enforce query to be rewritten for index access. This can e.g. be helpful if the name of a database is not static (see Enforce Rewritings for more examples):

(# db:enforceindex #) { for $db in ('persons1', 'persons2', 'persons3') return db:open($db)//name[text() = 'John'] }

• Temporarily disable node copying in node constructors (see COPYNODE for more details). The following query will be evaluated faster, and take much less memory, than without pragma, because the database nodes will not be fully copied, but only attached to the new xml parent element:

file:write( 'wrapped-db-nodes.xml', (# db:copynode false #) { <xml>{ db:open('huge') </xml> } )

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:

declare function local:square($a) { $a * $a }; for $i in 1 to 3 return local:square($i)

Query after function inlining:

for $i in 1 to 3 return
XQuery Extensions

let $a := $i
return $a * $a

Query after further optimizations:

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.

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().

Stack Trace:
- query.xq, 2/9

Lazy Evaluation

%basex:lazy enforces lazy evaluation of a global variable. An example:

Example:

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

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

XQuery Locks

Introduced with Version 9.1:

In analogy with option declarations and pragmas, XQuery Locks can also set via annotations:

declare %basex:write-lock('CONFIGLOCK') function local:write() {
  file:write('config.xml', <config/>)
};

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.
XQuery Extensions

let $n := 456
for $i in 1 to 2
return $n

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 $ignored := 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 $ignored. 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.

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. Module Library

Read this entry online in the BaseX Wiki.

This article is part of the XQuery Portal.

In addition to the standard XQuery Functions, BaseX comes with some hundred additional functions, which are packaged in various modules.

The namespaces of the following modules are statically known. This means that they need not be (but may be) declared in the query prolog.

<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 users.</td>
<td>admin</td>
<td><a href="http://basex.org/modules/admin">http://basex.org/modules/admin</a></td>
</tr>
<tr>
<td>Archive</td>
<td>Creating and processing archive ZIP archives.</td>
<td>archive</td>
<td><a href="http://basex.org/modules/archive">http://basex.org/modules/archive</a></td>
</tr>
<tr>
<td>Array</td>
<td>Functions for handling array arrays.</td>
<td>array</td>
<td><a href="http://www.w3.org/2005/xpath-functions/array">http://www.w3.org/2005/xpath-functions/array</a></td>
</tr>
<tr>
<td>Binary</td>
<td>Processing binary data.</td>
<td>bin</td>
<td><a href="http://expath.org/ns/binary">http://expath.org/ns/binary</a></td>
</tr>
<tr>
<td>Client</td>
<td>Executing commands and client queries on remote BaseX servers.</td>
<td>client</td>
<td><a href="http://basex.org/modules/client">http://basex.org/modules/client</a></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/modules/convert">http://basex.org/modules/convert</a></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/ns/crypto">http://expath.org/ns/crypto</a></td>
</tr>
<tr>
<td>CSV</td>
<td>Functions for processing CSV input.</td>
<td>csv</td>
<td><a href="http://basex.org/modules/csv">http://basex.org/modules/csv</a></td>
</tr>
<tr>
<td>Database</td>
<td>Functions for accessing db and updating databases.</td>
<td>db</td>
<td><a href="http://basex.org/modules/db">http://basex.org/modules/db</a></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/modules/fetch">http://basex.org/modules/fetch</a></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/ns/file">http://expath.org/ns/file</a></td>
</tr>
<tr>
<td>Full-Text</td>
<td>Functions for performing full-text operations.</td>
<td>ft</td>
<td><a href="http://basex.org/modules/ft">http://basex.org/modules/ft</a></td>
</tr>
<tr>
<td>Hashing</td>
<td>Cryptographic hash functions.</td>
<td>hash</td>
<td><a href="http://basex.org/modules/hash">http://basex.org/modules/hash</a></td>
</tr>
<tr>
<td>Higher-Order</td>
<td>Additional higher-order hof functions that are not in the standard libraries.</td>
<td>hof</td>
<td><a href="http://basex.org/modules/hof">http://basex.org/modules/hof</a></td>
</tr>
<tr>
<td>HTML</td>
<td>Functions for converting HTML input to XML documents.</td>
<td>html</td>
<td><a href="http://basex.org/modules/html">http://basex.org/modules/html</a></td>
</tr>
<tr>
<td>Module Library</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HTTP</strong></td>
<td>Sending HTTP requests, <a href="http://expath.org/ns/http-client">http</a> based on the EXPath HTTP module.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>Functions for requesting <a href="http://basex.org/modules/index">index</a> details on database indexes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Inspection</strong></td>
<td>Functions for extracting <a href="http://basex.org/modules/inspect">inspect</a> internal module information.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Jobs</strong></td>
<td>Organization of running <a href="http://basex.org/modules/jobs">jobs</a> commands and queries.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>JSON</strong></td>
<td>Parsing and serializing <a href="http://basex.org/modules/json">json</a> documents.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Lazy</strong></td>
<td>Functions for handling <a href="http://basex.org/modules/lazy">lazy</a> items.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Map</strong></td>
<td>Functions for handling <a href="http://www.w3.org/2005/xpath-functions/map">map</a> maps (key/value pairs).</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Math</strong></td>
<td>Mathematical operations, extending the W3C <a href="http://www.w3.org/2005/xpath-functions/math">math</a> Working Draft.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Functions for simplifying <a href="http://basex.org/modules/out">out</a> formatted output.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Executing system <a href="http://basex.org/modules/proc">proc</a> commands from XQuery.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Profiling</strong></td>
<td>Functions for profiling <a href="http://basex.org/modules/prof">prof</a> code snippets.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td>Functions for creating random <a href="http://basex.org/modules/random">random</a> numbers.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Repository</strong></td>
<td>Installing, deleting and <a href="http://basex.org/modules/repo">repo</a> listing packages.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>JDBC bridge to access <a href="http://basex.org/modules/sql">sql</a> relational databases.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Strings</strong></td>
<td>Functions for performing <a href="http://basex.org/modules/strings">strings</a> string computations.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td>Unit testing framework. <a href="http://basex.org/modules/unit">unit</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Update</strong></td>
<td>Functions for performing <a href="http://basex.org/modules/update">update</a> updates.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>User</strong></td>
<td>Creating and administering <a href="http://basex.org/modules/user">user</a> database users.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Utility</strong></td>
<td>Various utility and helper <a href="http://basex.org/modules/util">util</a> functions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Validation</strong></td>
<td>Validating documents: <a href="http://basex.org/modules/validate">validate</a> DTDs, XML Schema, RelaxNG.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Web</strong></td>
<td>Convenience functions for <a href="http://basex.org/modules/web">web</a> building web applications.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Module Library

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Prefix</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XQuery</strong></td>
<td>Evaluating new XQuery expressions at runtime.</td>
<td><code>xquery</code></td>
<td><a href="http://basex.org/modules/xquery">http://basex.org/modules/xquery</a></td>
</tr>
<tr>
<td><strong>XSLT</strong></td>
<td>Stylesheet transformations, based on Java’s and Saxon’s XSLT processor.</td>
<td><code>xslt</code></td>
<td><a href="http://basex.org/modules/xslt">http://basex.org/modules/xslt</a></td>
</tr>
<tr>
<td><strong>ZIP</strong></td>
<td>ZIP functionality, based on the EXPath ZIP module.</td>
<td><code>zip</code></td>
<td><a href="http://expath.org/ns/zip">http://expath.org/ns/zip</a></td>
</tr>
</tbody>
</table>

Some additional external modules exist, which can be used as follows:

- The `baseX-api` package must be included in the classpath. This is automatically the case if you use one of the complete distributions (zip, exe, war) of BaseX.

- The modules must be explicitly imported in the query prolog.

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Prefix</th>
<th>Namespace URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geo</td>
<td>Functions for processing geospatial data.</td>
<td><code>geo</code></td>
<td><a href="http://expath.org/ns/geo">http://expath.org/ns/geo</a></td>
</tr>
<tr>
<td>Request</td>
<td>Server-side functions for handling HTTP Request data.</td>
<td><code>request</code></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><code>restxq</code></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><code>session</code></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><code>sessions</code></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><code>ws</code></td>
<td><a href="http://basex.org/modules/ws">http://basex.org/modules/ws</a></td>
</tr>
</tbody>
</table>
Chapter 26. 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.

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 **java**: See the following examples:

- java:java.lang.String → java.lang.String

Functions and Variables

Java 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 (·, 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.

<table>
<thead>
<tr>
<th>Type</th>
<th>XQuery</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Q{java.lang.Integer}MIN_VALUE()</td>
<td>Integer.MIN_VALUE</td>
</tr>
<tr>
<td>Function</td>
<td>Q{java.lang.Object}hashCode()</td>
<td>object.hashCode()</td>
</tr>
<tr>
<td>Function with types</td>
<td>Q{java.lang.String}split</td>
<td>java.lang.String.split(';',''3)</td>
</tr>
</tbody>
</table>

As XQuery and Java have different type systems, XQuery arguments are converted to equivalent Java values, and the result of a Java function is converted back to an XQuery value (see Data Types).

If a Java function is not found, XQuery values may need to be cast the target type. For example, if a Java function expects a primitive `int` value, you will need to convert your XQuery integers to `xs:int`.

Namespace Declarations

In the following example, Java’s Math class is referenced. When executed, the query returns the cosine of an angle by calling the static method `cos()`, and the value of $\pi$ by addressing the static variable via `PI()`:

```java
Math.cos(angle) * Math.PI
```
With the *Expanded QName* notation of XQuery 3.0, the namespace can directly be embedded in the function call:

\[
\text{Q\{java:java.lang.Math\}cos(xs:double(0))}
\]

The constructor of a class can be invoked by calling the virtual function `new()`. Instance methods can then be called by passing on the resulting Java object as first argument. In the following example, 256 bytes are written to the file `output.txt`. First, a new `FileWriter` instance is created, and its `write()` function is called in the next step:

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

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

```xml
declare namespace br = 'java.io.BufferedReader';
declare namespace fr = 'java.io.FileReader';
declare option db:checkstrings 'false';
file:write-binary('00.bin', xs:hexBinary('00')),
br:new(fr:new('00.bin')) ! (br:readLine(.), br:close(.))
```

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

**Module Imports**

An alternative solution is to access Java code by importing classes as modules. A new instance of the addressed class will be created, which can then be referenced in the query body.

The following (side-effecting) example returns the number of distinct values added to a hash set. The boolean values returned by `set:add()` will be swallowed:

```xml
import module namespace set = "java:java.util.HashSet";
prof:void(
  for $s in ("one", "two", "one")
    return set:add($s)
),
set:size()
```

The advantages of this approach is the execution of imported classes is more efficient than the execution of instances that are created at runtime via `new()`. A drawback is that no arguments can be passed on to the class constructor. As a consequence, the import fails if the addressed class has no default constructor, but at least one constructor with arguments.

**Integration**

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

### Annotations

The internal properties of functions can be assigned via annotations:

- Java functions can only be executed by users with **Admin permissions**. You may annotate a function with `@Requires(<Permission>)` to also make it accessible to users with less privileges.

- Java code is treated as **non-deterministic**, as its behavior cannot be predicted by the XQuery processor. You may annotate a function as `@Deterministic` if you know that it will have no side-effects and will always yield the same result.

- Java code is treated as **context-independent**. If a function accesses the query context, it should be annotated as `@ContextDependent`

- Java code is treated as **focus-independent**. If a function accesses the current context item, position or size, it should be annotated as `@FocusDependent`

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

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

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

The imported Java class is shown below:

```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
   */
  @Requires(Permission.NONE)
  @Deterministic
  @ContextDependent
  public String user() {
    return queryContext.context.user.name;
  }

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

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

The result will look as follows:

<user>admin</admin>
<to-int>Error in line 6</to-int>

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

**Locking**

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

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

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

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

**Data Types**

The following table lists the mappings of XQuery and Java types:

<table>
<thead>
<tr>
<th>XQuery Type</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:string</td>
<td>String, char, Character</td>
</tr>
<tr>
<td>xs:boolean</td>
<td>boolean, Boolean</td>
</tr>
<tr>
<td>xs:byte</td>
<td>byte, Byte</td>
</tr>
<tr>
<td>xs:short</td>
<td>short, Short</td>
</tr>
<tr>
<td>xs:int</td>
<td>int, Integer</td>
</tr>
<tr>
<td>xs:long</td>
<td>long, Long</td>
</tr>
<tr>
<td>xs:float</td>
<td>float, Float</td>
</tr>
</tbody>
</table>
Java Bindings

<table>
<thead>
<tr>
<th>xs:double</th>
<th>double, Double</th>
</tr>
</thead>
<tbody>
<tr>
<td>xs:decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>xs:integer</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>xs:QName</td>
<td>javax.xml.namespace.QName</td>
</tr>
<tr>
<td>xs:anyURI</td>
<td>java.net.URI, java.net.URL</td>
</tr>
</tbody>
</table>

empty sequence null

**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. The following examples show some rewritings:

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

**Changelog**

Version 8.4
- 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 27. 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';
m:hello("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:

```bash
REPO INSTALL http://files.basex.org/modules/expatx/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:

```bash
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` sub-directory of the repository):

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

Importing the repository module:

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

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

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

Contents of the file Hello.java (comments removed):

```java
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):

```xml
import module namespace m = 'http://basex.org/modules/Hello';
```

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. If the package will be installed, the additional library archives will be extracted and copied to a hidden sub-directory in the repository. If the package will be deleted, the hidden sub-directory will be removed as well.

Examplary contents of Image.jar

```java
lib/
   Images.jar
META-INF/
   MANIFEST.MF
org/basex/modules/
   Image.class
```
Repository

Directory structure of the repository directory after installing the package

```
org/basex/modules/
  Image.class
  .Images/
    Images.jar
```

**Combined**

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`

```
import module namespace m = 'http://basex.org/modules/Hello';

m:hello("Universe")
```

**Wrapper Module** `Hello.xqm`

```
module namespace hello = 'http://basex.org/modules/Hello';

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`
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:

REPO INSTALL http://files.basex.org/modules/org/basex/modules/Hello.jar
REPO LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Type</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.basex.modules.Hello</td>
<td>-</td>
<td>Combined</td>
<td>org/basex/modules/Hello.xqm</td>
</tr>
</tbody>
</table>

EXPath Packaging

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

XQuery

Apart from the package descriptor, a .xar archive contains a directory which includes the actual XQuery modules. For example, the FunctX XAR archive is packaged as follows:

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

Java

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():
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:

```xquery
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 28. Full-Text

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

Full-text retrieval in XML documents is an essential requirement in many use cases. BaseX was the first query processor that supported the W3C XQuery Full Text 1.0 Recommendation, and it additionally comes with a powerful Full-Text Index, which allows you to evaluate full-text queries 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. 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 *ftand*, *ftor* and *ftnot* can also be used to combine multiple query terms. The following query yields the same result as the last one does:

(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:

```xml
'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:javadoc.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\xa;).
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: 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:

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

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

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

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

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

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

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

### Thesaurus

BaseX supports full-text queries using thesauri, but it does not provide a default thesaurus. This is why queries such as:
'computers' contains text 'hardware'
using thesaurus default

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

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

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

**Fuzzy Querying**

In addition to the official recommendation, BaseX supports 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, preserving a minimum of 1 errors. A static error distance can be set by adjusting the LSERROR property (default: SET LSERROR 0). The query above yields two results as there is no error between the query term “house” and the text node “house”, and one error between “house” and “hous”.

Fuzzy search is also supported by the full-text index.

**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:

- Turn off whitespace chopping when importing XML documents. This can be done by setting the option CHOP to OFF. This can also be done in the GUI if a new database is created (Database → New... → Parsing → Chop Whitespaces).
• Turn off automatic indentation by assigning indent=no to the SERIALIZER option.

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 (see Full-Text Functions) will only yield useful results if they are applied to single text nodes, as the following example demonstrates:

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

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 do not want to search for. See the following example (visit XQuery Update to learn more about updates):

let $docs := db:open('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: de, en-US.</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>
</tbody>
</table>
• “Ä” and “ä” are secondary differences,
• “Â” and “A&#308;” (see http://www.fileformat.info/info/unicode/char/308/index.htm) are tertiary differences, and
• “A” and “A” are identical.

decomposition Defines how composed characters are handled. Three decompositions are supported: none, standard, and full. More details are found in the JavaDoc of the JDK.

Some Examples:

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

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

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

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

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

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

Changelog

Version 8.0

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

Version 7.7

• Added: Collations support.

Version 7.3

• Removed: Trie index, which was specialized on wildcard queries. The fuzzy index now supports both wildcard and fuzzy queries.

• Removed: TF/IDF scoring was discarded in favor of the internal scoring model.
Chapter 29. Full-Text: Japanese

Read this entry online in the BaseX Wiki.

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

Introduction

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

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

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

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

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

Lexical Analysis

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

```
#  ##,##,##,##,##,##,##
#  ##,##,##,##,##,##,##
#  ##,##,##,##,##,##,##
##  ##,##,##,##,##,##,##
##  ##,##,##,##,##,##,##
##  ##,##,##,##,##,##,##
```

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

```
POS, subtyping POS 1, subtyping POS 2, subtyping POS 3, inflections, use type, prototype, reading, pronunciation
```

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

Parsing

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

- Mark

143
• Filler
• Postpositional particle
• Auxiliary verb

Thus, in the example above, #, #, and ## will be passed to the indexer for each token.

**Token Processing**

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

**Stemming**

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

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

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

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

'#####' contains text '##' using stemming using language 'ja'
'#####' contains text '###' using stemming using language 'ja'

**Wildcards**

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

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

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

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

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

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

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

'#####' contains text '## ftand .##' using wildcards using language 'ja'
Chapter 30. XQuery Update

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.

**Non-Updating Expressions**

**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:open('example')//entry)[1]
...

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

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

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

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

- More than one node can be specified as source:

```xml
db:open('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.
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**

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

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

It may be surprising to see `db:create` in the lower part of this list. This means that newly created database cannot be accessed by the same query, which can be explained by the semantics of updating queries: all expressions can only be evaluated on databases that already exist while the query is evaluated. As a consequence, `db:create` is mainly useful in the context of Command Scripts, or Web Applications, in which a redirect to another page can be triggered after having created a database.

**Example**

The query...

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

...applied on the document...

```xml
<doc> <a/> </doc>
```
…results in the following document:

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

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

**Returning Results**

By default, it is not possible to mix different types of expressions in a query result. The outermost expression of a query must either be a collection of updating or non-updating expressions. But there are two ways out:

- The BaseX-specific `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:

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

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

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

**Effects**

**Original Files**

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

- Updates on main-memory instances of files that have been retrieved via `fn:doc` or `fn:collection` will be propagated back to disk when the `WRITEBACK` option is turned on. This option can also be activated on command line via `-u`. Make sure you back up the original documents before running your queries.

- Functions like `fn:put` or `file:write` can be used to write single XML documents to disk. With `file:write-binary`, you can write binary resources.

- The `EXPORT` command can be used write all resources of a databases to disk.

**Indexes**

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

**Error Messages**

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

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 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 Applying text index for "Japan" indicates that the text index is applied to speed up the search of the shown string. The following message…

• Removing path with no index results indicates that a string in a path expression will never yield results. Because of that, 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.

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:

```
(: 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:

```
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:
Indexes

\[
\text{count(doc('factbook')//country)}
\]

- The distinct values of elements or attributes can be looked up in the index as well:

\[
\text{distinct-values(db:open('factbook')//religions)}
\]

The contents of the path index can be directly accessed with the XQuery function \text{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 values 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 \text{CREATE INDEX} and \text{DROP INDEX} are used to create and drop index structures. With \text{INFO INDEX}, you get some insight into the contents of an index structure, and \text{SET} allows you to change the index defaults for new databases:

- \text{OPEN factbook; CREATE INDEX fulltext} : Open database; create full-text index
- \text{OPEN factbook; INFO INDEX TOKEN} : Open database; show info on token index
- \text{SET ATTRINDEX true; SET ATTRINCLUDE id name} ; \text{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 \text{db:optimize}:

\[
\text{db:optimize('factbook', false(), map { 'ftindex': true(), 'ftinclude': 'p div'} )}
\]

Text Index

Exact Queries

This index speeds up string-based equality tests on text nodes. The following queries will all be rewritten for index access:

\[
\text{1st example :}
\]
\[
//*[text() = 'Germany']
\]
\[
\text{2nd example :}
\]
\[
doc('factbook.xml')//name[. = 'Germany']
\]
\[
\text{3rd example :}
\]
\[
for $c in db:open('factbook')//country where $c//city/name = 'Hanoi'
\]
Indexes

Matching text nodes can be directly requested from the index with the XQuery function `db:text`. The index contents can be accessed via `index:text`.

The `UPDINDEX` option can be activated to keep this index up-to-date, for example:

```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:open('Library')//Medium[Year >= '2011' and Year <= '2016']`,
2. `let $min := '2014-04-16T00:00:00'
   return db:open('news')//entry[date-time > $min and date-time < $max]
```

Text nodes can directly be retrieved from the index via the XQuery function `db:text-range`.

Please note that the current index structures do not support queries for numbers and dates.

Attribute Index

Similar to the text index, this index speeds up string-based equality and range tests on attribute values. 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:open('factbook'))`

Attribute nodes can directly be retrieved from the index with the XQuery functions `db:attribute` and `db:attribute-range`. The index contents 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. `fn:idref('f0_119', db:open('factbook'))`
Attributes with tokens can be directly retrieved from the index with the XQuery function `db:token`. The index contents 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:

```xquery
//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: ST marginal)
- **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:

```xml
<commands>
  <!-- Create database with stemmed full-text index -->
  <set option='stemming'>true</set>
  <set option='ftindex'>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
Indexes

- **name**: elements or attributes called `name`, which are in the empty default namespace
- `*:name`: elements or attributes called `name`, no matter which namespace
- `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 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' || $n
  return db:open($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:open('factbook')//country
  [(@population > '10000000' and @population < '10999999')
  [religions/text() = 'Protestant']
  ```

With Version 9.1, 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:open('factbook')//country
where (# db:enforceindex #) { $country/name = $name }
where $country/religions/text() = 'Protestant'
return $country

Please note that:

- 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).
- The enforcement of index rewritings gives you no guarantee that an index will be used. There are many expressions that cannot be rewritten for index access.
- If you have more than one enforce pragma in a single path expression, only the first will be considered.

### Custom Index Structures

With XQuery, it is comparatively easy to create your own, custom index structures. The following query demonstrate how you can create a `factbook-index` database, which contains all texts of the original database in lower case:

```xquery
let $db := 'factbook'
let $index := <index>
  for $nodes in db:open($db)//text()
  group by $text := lower-case($nodes)
  return <text string='( $text )'><!--
    for $node in $nodes
    return <id>{ db:node-id($node ) }-->
  </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:

```xquery
let $db := 'factbook'
let $text := 'italian'
for $id in db:open($db || '-index')//*[@string = $text]/id
return db:open-id($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 to 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:
Indexes

> 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.
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 is 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 meta data 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

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 are textually output, i.e., serialized. (For input, see Parsers.) They have been formalized in the W3C XQuery Serialization 3.1 document. In BaseX, they can be specified by...

- including them in the prolog of the XQuery expression,
- specifying them in the XQuery functions file:write() or fn:serialize(). The serialization parameters are specified as
  - children of an <output:serialization-parameters/> element, as defined for the fn:serialize() function, or as
  - map, which contains all key/value pairs: map { "method": "xml", "cdata-section-elements": "div", ... },
- using the -s flag of the BaseX command-line clients,
- setting the SERIALIZER option before running a query,
- setting the EXPORTER option before exporting a database, or
- setting them as REST query parameters.

Parameters

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

<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. xml, xhtml, html, text, json, and adaptive are adopted from the official specification. The methods basex and csv are specific to BaseX (see XQuery Extensions).</td>
<td>xml, xhtml, html, text, json, adaptive, csv, basex</td>
<td>basex</td>
</tr>
<tr>
<td>version</td>
<td>Specifies the version of the serialization method.</td>
<td>xml/xhtml: 1.0, 1.1html: 4.0, 4.01, 5.0</td>
<td>1.0</td>
</tr>
<tr>
<td>html-version</td>
<td>Specifies the version of the HTML serialization method.</td>
<td>4.0, 4.01, 5.0</td>
<td>4.0</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>arbitrary strings</td>
<td>empty</td>
</tr>
<tr>
<td>encoding</td>
<td>Encoding to be used for outputting the data. all encodings supported by Java</td>
<td>UTF-8 Java</td>
<td>UTF-8</td>
</tr>
<tr>
<td>indent</td>
<td>Adjusts whitespaces to make the output better readable.</td>
<td>yes, no</td>
<td>yes</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cdata-section-elements</td>
<td>List of elements to be output as CDATA, separated by whitespaces. Example: <code>&lt;text&gt; &lt;![CDATA[ &lt;&gt; ]] &lt;/text&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>omit-xml-declaration</td>
<td>Omits the XML declaration, which is serialized before the actual query result. Example: <code>&lt;?xml version=&quot;1.0&quot; encoding=&quot;UTF-8&quot;?&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>standalone</td>
<td>Prints or omits the &quot;standalone&quot; attribute in the XML declaration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>doctype-system</td>
<td>Introduces the output with a document type declaration and the given system identifier. Example: <code>&lt;!DOCTYPE x SYSTEM &quot;entities.dtd&quot;&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doctype-public</td>
<td>If doctype-system is specified, adds a public identifier. Example: <code>&lt;!DOCTYPE HTML PUBLIC &quot;-//W3C//DTD HTML 4.01//EN&quot;  &quot;http://www.w3.org/TR/html4/strict.dtd&quot;&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>undeclare-prefixes</td>
<td>Undeclares prefixes in XML 1.1.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>normalization-form</td>
<td>Specifies a normalization form. BaseX supports Form C (NFC).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>media-type</td>
<td>Specifies the media type.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameter-document</td>
<td>Parses the value as XML document with additional serialization parameters (see the Serialization Specification for more details).</td>
<td></td>
<td></td>
</tr>
<tr>
<td>use-character-maps</td>
<td>Defines character mappings. May only occur in documents parsed with parameter-document.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte-order-mark</td>
<td>Prints a byte-order-mark before starting serialization.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>escape-uri-attributes</td>
<td>Escapes URI information in certain HTML attributes. Example: <code>&lt;a href=&quot;%C3%A4%C3%B6%C3%BC&quot;&gt;äöü&lt;/a&gt;</code></td>
<td></td>
<td></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: <code>&lt;head&gt;&lt;meta http-</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Serialization

```html
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></head>
```

The head element must already exist or nothing will be added. Any existing meta content-type elements will be removed.

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 ,:

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

If `fn:serialize` is called, output-specific parameters can be supplied via nested options:

```
serialize(
```
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 &nbsp; (without the serialization parameter, the Unicode character would be output):

Example query:

```
declare option output:parameter-document "map.xml";
<x>&#xA0;</x>
```

Example parameter-document:

```
<serialization-parameters
 xmlns="http://www.w3.org/2010/xslt-xquery-serialization">
 <use-character-maps>
   <character-map character="&#160;" map-string="&amp;#160;"/>
 </use-character-maps>
</serialization-parameters>
```

Changelog

Version 8.4

- Added: Serialization parameter binary.

- Updated: New serialization method base64. 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, json-unescape, json-spec, json-format.

Version 7.5
• Added: official item-separator and html-version parameter.

• Updated: method=html5 removed; serializers updated with the latest version of the specification, using method=html and version=5.0.

Version 7.2
• Added: separator parameter.

Version 7.1
• Added: newline parameter.

Version 7.0
• Added: Serialization parameters added to REST API; JSON/JsonML/raw methods.
Chapter 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 EXPat HTTP Specifications.

**Static Errors**

- Namespace URI: 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., 1+ for i in //* return $i 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>1+ for i in //* return $i</td>
</tr>
<tr>
<td>XPST0005</td>
<td>An expression will never return any results, no matter what input is provided.</td>
<td>doc('input')/..</td>
</tr>
<tr>
<td>XPST0008</td>
<td>A variable or type name is used that has not been defined in the current scope.</td>
<td>$a--- element (*, x)</td>
</tr>
<tr>
<td>XQST0003</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>unknown() count (1,2,3)</td>
</tr>
<tr>
<td>XQST00080</td>
<td>xs:NOTATION or xs:anyAtomicType is used as castable as xs:NOTATION target type of cast or castable.</td>
<td>1 castable as xs:itr</td>
</tr>
<tr>
<td>XQST0081</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>unknown:x (# pragma #) { 1 }</td>
</tr>
<tr>
<td>XQST0009</td>
<td>The query imports a schema (schema import is not supported by BaseX).</td>
<td>&lt;elem xmlns=&quot;( 'dynamic' )&quot;/&gt;</td>
</tr>
<tr>
<td>XQST0022</td>
<td>Namespace values must be constant strings.</td>
<td>&lt;elem xmlns=&quot;( 'dynamic' )&quot;/&gt;</td>
</tr>
<tr>
<td>XQST0031</td>
<td>The specified XQuery version is not specified.</td>
<td>xquery version &quot;9.9&quot;; ()</td>
</tr>
<tr>
<td>XQST0032</td>
<td>The base URI was declared more than once.</td>
<td>declare base-uri ...</td>
</tr>
<tr>
<td>XQST0033</td>
<td>A namespace prefix was declared more than once.</td>
<td>declare a=&quot;a&quot;; declare a=&quot;b&quot;; ()</td>
</tr>
<tr>
<td>XQST0034</td>
<td>A function was declared more than once.</td>
<td>declare function local:a() { 1 }; declare function local:a() { 2 }; local:a()</td>
</tr>
</tbody>
</table>

164
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>XQST0038</td>
<td>The default collation was declared more than once.</td>
<td><code>declare default collation ...</code></td>
</tr>
<tr>
<td>XQST0039</td>
<td>Two or more parameters in a user-defined function have the same name.</td>
<td><code>declare function local:fun($a, $a) { $a * $a };local:fun(1,2)</code></td>
</tr>
<tr>
<td>XQDY0040</td>
<td>Two or more attributes in an element have the same node name.</td>
<td><code>&lt;elem a=&quot;1&quot; a=&quot;12&quot;/&gt;</code></td>
</tr>
<tr>
<td>XQDY0045</td>
<td>A user-defined function uses a reserved namespace.</td>
<td><code>declare function fn:fun() { 1 };()</code></td>
</tr>
<tr>
<td>XQST0047</td>
<td>A module was defined more than once.</td>
<td><code>import module ...</code></td>
</tr>
<tr>
<td>XQST0048</td>
<td>A module declaration does not match the namespace of the specified module.</td>
<td><code>import module namespace invalid=&quot;uri&quot;; 1</code></td>
</tr>
<tr>
<td>XQST0049</td>
<td>A global variable was declared more than once.</td>
<td><code>declare variable $a := 1; declare variable $a := 1; $a</code></td>
</tr>
<tr>
<td>XQST0054</td>
<td>A global variable depends on itself. This may be triggered by a circular variable definition.</td>
<td><code>declare variable $a := local:a(); declare function local:a() { $a }; $a</code></td>
</tr>
<tr>
<td>XQST0055</td>
<td>The mode for copying namespaces was declared more than once.</td>
<td><code>declare copy-namespaces ...</code></td>
</tr>
<tr>
<td>XQST0057</td>
<td>The namespace of a schema import may not be empty.</td>
<td><code>import schema &quot;&quot;; ()</code></td>
</tr>
<tr>
<td>XQST0059</td>
<td>The schema or module with the specified namespace cannot be found or processed.</td>
<td><code>import module &quot;unknown&quot;; ()</code></td>
</tr>
<tr>
<td>XQST0060</td>
<td>A user-defined function has no namespace.</td>
<td><code>declare default function namespace &quot;&quot;; declare function x() { 1 }; 1</code></td>
</tr>
<tr>
<td>XQST0065</td>
<td>The ordering mode was declared more than once.</td>
<td><code>declare ordering ...</code></td>
</tr>
<tr>
<td>XQST0066</td>
<td>The default namespace mode for elements or functions was declared more than once.</td>
<td><code>declare default element namespace ...</code></td>
</tr>
<tr>
<td>XQST0067</td>
<td>The construction mode was declared more than once.</td>
<td><code>declare construction ...</code></td>
</tr>
<tr>
<td>XQST0068</td>
<td>The mode for handling boundary spaces was declared more than once.</td>
<td><code>declare boundary-space ...</code></td>
</tr>
<tr>
<td>XQST0069</td>
<td>The default order for empty sequences was declared more than once.</td>
<td><code>declare default order empty ...</code></td>
</tr>
<tr>
<td>XQST0070</td>
<td>A namespace declaration overwrites a reserved namespace.</td>
<td><code>declare namespace xml=&quot;&quot;; ()</code></td>
</tr>
<tr>
<td>XQST0071</td>
<td>A namespace is declared more than once in an element constructor.</td>
<td><code>&lt;a xmlns=&quot;uri1&quot; xmlns=&quot;uri2&quot;/&gt;</code></td>
</tr>
<tr>
<td>XQST0075</td>
<td>The query contains a validate expression (validation is not supported by BaseX).</td>
<td><code>(validate strict { () })</code></td>
</tr>
<tr>
<td>XQST0076</td>
<td>A group by or order by clause specifies an unknown collation.</td>
<td><code>(for $i in 1 to 10 order by $i collation &quot;unknown&quot; return $i)</code></td>
</tr>
<tr>
<td>XQST0079</td>
<td>A pragma was specified without the expression that is to be evaluated.</td>
<td><code>(# xml:a #) ()</code></td>
</tr>
<tr>
<td>XQST0085</td>
<td>An empty namespace URI was specified.</td>
<td><code>&lt;pref:elem xmlns:pref=&quot;&quot;/&gt;</code></td>
</tr>
<tr>
<td>XQST0087</td>
<td>An unknown encoding was specified. Note that the encoding declaration is currently ignored in BaseX.</td>
<td><code>xquery version &quot;1.0&quot; encoding &quot;a b&quot;; ()</code></td>
</tr>
<tr>
<td>XQST0088</td>
<td>An empty module namespace was specified.</td>
<td><code>import module &quot;&quot;; ()</code></td>
</tr>
<tr>
<td>XQST0089</td>
<td>Two variables in a for or let clause have the same name.</td>
<td><code>for $a at $a in 1 return $i</code></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.

XQST0097 A decimal-format property is invalid.

XQST0098 A single decimal-format character was assigned to multiple properties.

XQST0099 The context item was declared more than once.

XQST0106 An annotation has been declared twice in a variable or function declaration.

XQST0108 Output declarations may only be specified in the main module.

XQST0109 The specified serialization parameter is unknown.

XQST0110 A serialization parameter was specified more than once in the output declarations.

XQST0111 A decimal format was declared more than once.

XQST0113 Context item values may only be in the main module.

XQST0114 A decimal-format property has been specified more than once.

Type Errors

- Namespace URI: http://www.w3.org/2005/xqt-errors
- Namespace prefix: err
- Codes: XPTY, XQTY

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPTY0004</td>
<td>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).</td>
<td>1 + &quot;A&quot; abs(&quot;a&quot;) 1 cast as xs:gYear</td>
</tr>
<tr>
<td>XPTY0018</td>
<td>The result of the last step in a path expression contains both nodes and atomic values.</td>
<td>doc('input.xml')/(*, 1)</td>
</tr>
<tr>
<td>XPTY0019</td>
<td>The result of a step (other than the last step) in a path expression contains an atomic values.</td>
<td>(1 to 10)/*</td>
</tr>
<tr>
<td>XQTY0024</td>
<td>An attribute node cannot be bound to its parent element.</td>
<td>&lt;elem&gt;text { attribute a as other nodes of a different type were specified before. ( &quot;val&quot; } &lt;/elem&gt;</td>
</tr>
<tr>
<td>XQTY0105</td>
<td>A function item has been specified as content of an element.</td>
<td>&lt;X&gt;{ false#0 }&lt;/X&gt;</td>
</tr>
</tbody>
</table>

Dynamic Errors

- Namespace URI: http://www.w3.org/2005/xqt-errors
- Namespace prefix: err
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; $x descendant::*</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. | treat xs:int as "string" /*

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

Functions Errors

• 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>12345678901234567890 $a := (1,2) group by $a return $a</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:double(&quot;INF&quot;) idiv 1 xs:int(xs:double(&quot;NaN&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>
<tr>
<td>Error Code</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>FOCH0001</td>
<td>A codepoint was specified that does not represent a valid XML character.</td>
<td></td>
</tr>
<tr>
<td>FOCH0002</td>
<td>An unsupported collation was specified in a function.</td>
<td></td>
</tr>
<tr>
<td>FOCH0003</td>
<td>An unsupported normalization form was specified in a function.</td>
<td></td>
</tr>
<tr>
<td>FODC0001</td>
<td>The argument specified in <code>fn:id()</code> or <code>fn:idref()</code> must have a document node as root.</td>
<td></td>
</tr>
<tr>
<td>FODC0002</td>
<td>The specified document resource cannot be retrieved.</td>
<td></td>
</tr>
<tr>
<td>FODC0004</td>
<td>The specified collection cannot be retrieved.</td>
<td></td>
</tr>
<tr>
<td>FODC0005</td>
<td>The specified URI to a document resource is invalid.</td>
<td></td>
</tr>
<tr>
<td>FODC0006</td>
<td>The string passed to <code>fn:parse-xml()</code> is not well-formed.</td>
<td></td>
</tr>
<tr>
<td>FODC0007</td>
<td>The base URI passed to <code>fn:parse-xml()</code> is invalid.</td>
<td></td>
</tr>
<tr>
<td>FODF1280</td>
<td>The name of the decimal format passed to <code>fn:format-number()</code> is invalid.</td>
<td></td>
</tr>
<tr>
<td>FODF1310</td>
<td>The picture string passed to <code>fn:format-number()</code> is invalid.</td>
<td></td>
</tr>
<tr>
<td>FODT0001</td>
<td>An arithmetic duration operation causes an over- or underflow.</td>
<td></td>
</tr>
<tr>
<td>FODT0002</td>
<td>A duration declaration or operation causes an over- or underflow.</td>
<td></td>
</tr>
<tr>
<td>FODT0003</td>
<td>An invalid timezone was specified.</td>
<td></td>
</tr>
<tr>
<td>FOER0000</td>
<td>Error triggered by the <code>fn:error()</code> function.</td>
<td></td>
</tr>
<tr>
<td>FOFD1340</td>
<td>The picture string passed to <code>fn:format-date()</code>, <code>fn:format-time()</code> or <code>fn:format-date()</code> is invalid.</td>
<td></td>
</tr>
<tr>
<td>FOFD1350</td>
<td>The picture string passed to <code>fn:format-time()</code> specifies an non-available component.</td>
<td></td>
</tr>
<tr>
<td>FONS0004</td>
<td>A function has a QName as argument that specifies an unbound prefix.</td>
<td></td>
</tr>
<tr>
<td>FORG0001</td>
<td>A value cannot be cast to the required target type.</td>
<td></td>
</tr>
<tr>
<td>FORG0002</td>
<td>The URI passed to <code>fn:resolve-URI()</code> is invalid.</td>
<td></td>
</tr>
<tr>
<td>FORG0003</td>
<td><code>fn:zero-or-one()</code> was called with more than one item.</td>
<td></td>
</tr>
<tr>
<td>FORG0004</td>
<td><code>fn:one-or-more()</code> was called with zero items.</td>
<td></td>
</tr>
<tr>
<td>FORG0005</td>
<td><code>fn:exactly-one()</code> was called with zero or more than one item.</td>
<td></td>
</tr>
<tr>
<td>FORG0006</td>
<td>A wrong argument type was specified in a function call.</td>
<td></td>
</tr>
<tr>
<td>FORG0008</td>
<td>The arguments passed to <code>fn:dateTime()</code> have different timezones.</td>
<td></td>
</tr>
<tr>
<td>FORX0001</td>
<td>A function specifies an invalid regular expression flag.</td>
<td></td>
</tr>
</tbody>
</table>
XQuery Errors

FORX0002 A function specifies an invalid regular expression.  
matches('input', '[')

FORX0003 A regular expression matches an empty string.  
tokenize('input', '.?')

FORX0004 The replacement string of a regular expression is invalid.  
replace("input", "match", ")"

FOTY0012 An item has no typed value.  
count#1

FOTY0013 Functions items cannot be atomized, have no defined equality, and have no string representation.
data(false#0)

FOTY0014 Function items have no string representation.  
string(map ())

FOTY0015 Function items cannot be compared.  
deep-equal(false#0, true#0)

FOUT1170 Function argument cannot be used to retrieve a text resource.  
unparsed-text(':')

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

Serialization Errors

- 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. fn:put(&lt;a/&gt;, '///')</td>
<td></td>
</tr>
</tbody>
</table>
### XQuery Errors

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XUDY0009</td>
<td>The target node of a replace expression needs a parent in order to be replaced.</td>
</tr>
<tr>
<td>XUDY0014</td>
<td>The expression updated by the modify clause was not created by the copy clause.</td>
</tr>
<tr>
<td>XUDY0015</td>
<td>In a rename expression, a target is renamed more than once.</td>
</tr>
<tr>
<td>XUDY0016</td>
<td>In a replace expression, a target is replaced more than once.</td>
</tr>
<tr>
<td>XUDY0017</td>
<td>In a replace value of expression, a target is replaced more than once.</td>
</tr>
<tr>
<td>XUDY0021</td>
<td>The resulting update expression contains duplicate attributes.</td>
</tr>
<tr>
<td>XUDY0023</td>
<td>The resulting update expression conflicts with existing namespaces.</td>
</tr>
<tr>
<td>XUDY0024</td>
<td>New namespaces conflict with each other.</td>
</tr>
<tr>
<td>XUDY0027</td>
<td>Target of an update expression is an empty sequence.</td>
</tr>
<tr>
<td>XUDY0029</td>
<td>The target of an update expression has no parent node.</td>
</tr>
<tr>
<td>XUDY0030</td>
<td>Attributes cannot be inserted before or after the child of a document node.</td>
</tr>
<tr>
<td>XUDY0031</td>
<td>Multiple calls to fn:put() address the same URI.</td>
</tr>
<tr>
<td>XUST0001</td>
<td>No updating expression is allowed here.</td>
</tr>
<tr>
<td>XUST0002</td>
<td>An updating expression is expected in the modify clause or an updating function.</td>
</tr>
<tr>
<td>XUST0003</td>
<td>The revalidation mode was declared more than once.</td>
</tr>
<tr>
<td>XUST0004</td>
<td>The query contains a revalidate expression (revalidation is not supported by BaseX).</td>
</tr>
<tr>
<td>XUST0005</td>
<td>No return type may be specified in an updating function.</td>
</tr>
<tr>
<td>XUTY0004</td>
<td>New attributes to be inserted must directly follow the root node.</td>
</tr>
<tr>
<td>XUTY0005</td>
<td>A single element or document node is expected as target of an insert expression.</td>
</tr>
</tbody>
</table>
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. delete node "string"

XUTY0008 A single element, text, attribute, comment or processing replace node document { <a/> } with <b/> instruction is expected as target of a replace > ) with <b/> expression.

XUTY0010 In a replace expression, in which no attributes are replace node <a><b/></a>/b targeted, the replacing nodes must not be attributes as with attribute size { 1 } well.

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/> element, attribute or processing instruction.

XUTY0013 An expression in the copy clause must return a single node. copy $c := (<a/>, <b/>modify(() return $c

XUTY0022 An attribute must not be inserted into a document node. insert node <e a=""/>/@a into document ('a')

Full-Text Errors

- 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>171</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 fn:doc 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 VI. XQuery Modules
Chapter 34. Admin Module

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.

Functions

admin:sessions

Signatures

admin:sessions() as element(session)*

Summary

Returns an element sequence with all currently opened database sessions, including the user name, address (IP:port) and an optionally opened database. The output of this function and the SHOW SESSIONS command is similar.

Examples

- admin:sessions() may e.g. return
  <session user="admin" address="127.0.0.1:6286" database="factbook"/>

admin:logs

Signatures

admin:logs() as element(file)*
admin:logs($date as xs:string) as element(entry)*
admin:logs($date as xs:string, $merge as xs:boolean) as element(entry)*

Summary

Returns Logging data compiled by the database or HTTP server:

- If no argument is specified, a list of all log files will be returned, including the file size and date.
- If a $date is specified, the contents of a single log file will be returned.
- If $merge is set to true, related log entries will be merged. Please note that the merge might not be 100% successful, as log entries may be ambiguous.

Examples

- admin:logs() may return
  <file size="834367"/>2015-01-23</file> if a single log file exists.
- admin:logs() ! admin:logs(.) lists the contents of all log files.

admin:write-log

Signatures

admin:write-log($text as xs:string) as empty-sequence()
admin:write-log($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, user name). An optional log $type can be specified, which must consist of letters in upper case. 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: Log type must consist of uppercase letters.

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.

date

<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>Log type must consist of uppercase letters.</td>
</tr>
</tbody>
</table>

Changelog

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.

Functions

archive:create

<table>
<thead>
<tr>
<th>Signatures</th>
<th>archive:create($entries as item(), $contents as item()*) as xs:base64Binary</th>
<th>archive:create($entries as item(), $contents as item()<em>, $options as map(</em>)?) as xs:base64Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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 <code>xs:string</code>, representing the entry name, or <code>element(archive:entry)</code>, containing the name as text node and additional, optional attributes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• last-modified: timestamp, specified as <code>xs:dateTime</code> (default: current time)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• compression-level: 0-9, 0 = uncompressed (default: 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• encoding: for textual entries (default: UTF-8)</td>
<td></td>
</tr>
<tr>
<td>The actual $contents must be <code>xs:string</code> or <code>xs:base64Binary</code> items. The $options parameter contains archiving options:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• format: allowed values are zip and gzip. zip is the default.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• algorithm: allowed values are deflate and stored (for the zip format). deflate is the default.</td>
<td></td>
</tr>
<tr>
<td>Errors</td>
<td>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 the CHECKSTRINGS option is turned off. single: the chosen archive format only allows single entries. error: archive creation failed for some other reason.</td>
<td></td>
</tr>
<tr>
<td>Examples</td>
<td>The following one-liner creates an archive archive.zip with one file file.txt:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>archive:create(<a href="">archive:entry</a>file.txt&lt;/archive:entry&gt;, 'Hello World')</td>
<td></td>
</tr>
</tbody>
</table>
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 ! element archive:entry { . },
    $files ! file:read-binary($path || .))
return file:write-binary('mp3.zip', $zip)
```

### archive:create-from

**Signatures**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>archive:create-from($path as xs:string) as xs:base64Binary</code></td>
<td>This convenience function creates an archive from all files in the specified directory <code>$path</code>. The <code>$options</code> parameter contains archiving options, and the files to be archived can be limited via <code>$entries</code>. The format of the two last arguments is the same as for <code>archive:create</code>.</td>
</tr>
</tbody>
</table>

**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 the same as for `archive:create`.

**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 for some other reason.

**Examples**

This example writes the files of a user's home directory to `archive.zip`:

```xml
let $zip := archive:create-from('/home/user/)
return file:write-binary('archive.zip', $zip)
```

### archive:entries

**Signatures**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>archive:entries($archive as xs:base64Binary) as element(archive:entry)*</code></td>
<td>Returns the entry descriptors of the specified <code>$archive</code>. A descriptor contains the following attributes, provided that they are available in the archive format:</td>
</tr>
</tbody>
</table>

- `size`: original file size
- `last-modified`: timestamp, formatted as `xs:dateTime`
- `compressed-size`: compressed file size

**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:

```xml
<archive:entry size="1840" last-modified="2009-03-20T03:30:32"
    compressed-size="672">
    doc/index.html
</archive:entry>
```

**Errors**

- `error`: archive creation failed for some other reason.

**Examples**

Sums up the file sizes of all entries of a JAR file:

```xml
sum(archive:entries(file:read-binary('zip.zip'))/@size)
```

### archive:options

**Signatures**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>archive:options($archive as xs:base64Binary) as map(*)</code></td>
<td></td>
</tr>
</tbody>
</table>
### Archive Module

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the options of the specified $archive in the format specified by archive:create.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>format: The packing format is not supported. error: archive creation failed for some other reason.</td>
</tr>
<tr>
<td>Examples</td>
<td>A standard ZIP archive will return the following options:</td>
</tr>
</tbody>
</table>

```
map {
  "format": "zip",
  "algorithm": "deflate"
}
```

### archive:extract-text

<table>
<thead>
<tr>
<th>Signatures</th>
<th>archive:extract-text($archive as xs:base64Binary) as xs:string* archive:extract-text($archive as xs:base64Binary, $entries as item()) as xs:string* archive:extract-text($archive as xs:base64Binary, $entries as item()), $encoding as xs:string) as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>encode: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the CHECKSTRINGS option is turned off. error: archive creation failed for some other reason.</td>
</tr>
<tr>
<td>Examples</td>
<td>The following expression extracts all .txt files from an archive:</td>
</tr>
</tbody>
</table>

```
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

<table>
<thead>
<tr>
<th>Signatures</th>
<th>archive:extract-binary($archive as xs:base64Binary) as xs:base64Binary* archive:extract-binary($archive as xs:base64Binary, $entries as item()) as xs:base64Binary*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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).</td>
</tr>
<tr>
<td>Errors</td>
<td>error: archive creation failed for some other reason.</td>
</tr>
<tr>
<td>Examples</td>
<td>This example unzips all files of an archive to the current directory:</td>
</tr>
</tbody>
</table>

```
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)
})
```

### archive:extract-to

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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()</th>
</tr>
</thead>
</table>
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 for some other reason.

Examples

The following expression unzips all files of an archive to the current directory:

archive:extract-to('.', file:read-binary('archive.zip'))

archive:update

Signatures

archive:update($archive as xs:base64Binary, $entries as item()*, $contents as item()*) as xs:base64Binary

Summary

Creates an updated version of the specified $archive with new or replaced entries. The format of $entries and $contents is the same as for archive:create.

Errors

number: the number of entries and contents differs.
descriptor: entry descriptors contain invalid entry names, timestamps, compression levels or encodings.
encode: the specified encoding is invalid or not supported, or the string conversion failed. Invalid XML characters will be ignored if the CHECKSTRINGS option is turned off.
modify: the entries of the given archive cannot be modified.
error: archive creation failed for some other reason.

Examples

This example replaces texts in a Word document:

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

archive:delete

Signatures

archive:delete($archive as xs:base64Binary, $entries as item()*) as xs:base64Binary

Summary

Deletes entries from an $archive. The format of $entries is the same as for archive:create.

Errors

modify: the entries of the given archive cannot be modified.
error: archive creation failed for some other reason.

Examples

This example deletes all HTML files in an archive and creates a new file:

```xml
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))
```

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 the CHECKSTRINGS option is turned off.</td>
</tr>
</tbody>
</table>
error Archive processing failed for some other reason.
format The packing format or the specified option is invalid or not supported.
modify The entries of the given archive cannot be modified.
number The number of specified entries and contents differs.
single The chosen archive format only allows single entries.

**Changelog**

Version 9.0

• 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)

Version 7.7

• Added: 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(*)

181
### Array Module

**Summary**

Returns a copy of `$array` with `$member` replaced at the specified `$position`. Equivalent to `$array => array:remove($position) => array:insert-before($position, $member)`.

**Errors**

FOAY0001: `$position` is not in the range 1 to `array:size($array)` inclusive.

**Examples**

```markdown
* array:put(["a", "b", "c"], 2, "d") returns the array ["a", "d", "c"].
```

### array:remove

**Signatures**

`array:remove($array as array(*), $positions as xs:integer*) as array(*)`

**Summary**

Returns a copy of `$array` without the member at the specified `$positions`.

**Errors**

FOAY0001: A position is not in the range 1 to `array:size($array)` inclusive.

**Examples**

```markdown
* array:append(["a"], 1) returns the array [].
```

### array:insert-before

**Signatures**

`array:insert-before($array as array(*), $position as xs:integer, $member as item(*)*) as array(*)`

**Summary**

Returns a copy of `$array` with one new `$member` at the specified `$position`. Setting `$position` to the value `array:size($array) + 1` yields the same result as `array:append($array, $insert)`.

**Errors**

FOAY0001: `$position` is not in the range 1 to `array:size($array) + 1` inclusive.

**Examples**

```markdown
* array:insert-before(["a"], 1, "b") returns the array ["b", "a"].
```

### array:head

**Signatures**

`array:head($array as array(*)) as item(*)`

**Summary**

Returns the first member of `$array`. This function is equivalent to the expression `$array(1)`.

**Errors**

FOAY0001: The array is empty.

**Examples**

```markdown
* array:head(["a", "b"]) returns "a".
  * array:head([["a", "b"], ["c", "d"]]) returns the array ["a", "b"].
```

### array:tail

**Signatures**

`array:tail($array as array(*)) as array(*)`

**Summary**

Returns a new array with all members except the first from `$array`. This function is equivalent to the expression `array:remove($array, 1)`.

**Errors**

FOAY0001: The array is empty.

**Examples**

```markdown
* array:insert-before(["a"], 1, "b") returns the array ["b", "a"].
```

### array:reverse

**Signatures**

`array:reverse($array as array(*)) as array(*)`  

**Summary**

Returns a new array with all members of `$array` in reverse order.

**Examples**

```markdown
* array:reverse(array { 1 to 3 }) returns the array [3, 2, 1].
```

### array:join

**Signatures**

`array:join($arrays as array(*))* as array(*)`
### Array Module

<table>
<thead>
<tr>
<th>Summary</th>
<th>Concatenates the contents of several $arrays into a single array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>• array:join(()) returns the array [].</td>
</tr>
<tr>
<td></td>
<td>• array:join((1 to 3) ! array { . }) returns the array [1, 2, 3].</td>
</tr>
</tbody>
</table>

#### array:flatten

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

#### array:for-each

| Signatures | array:for-each($array as array(*), $function as function(item()*) as item()*) as array(*) |
| Summary | Returns a new array, in which each member is computed by applying $function to the corresponding member of $array. |
| Examples | The following query returns the array [2, 3, 4, 5, 6]: |
|         | array:for-each( |
|         |   array ( 1 to 5 ), |
|         |   function($i) ( $i + 1) |
|         | ) |

#### array:filter

| Signatures | array:filter($array as array(*), $function as function(item()*) as xs:boolean) as array(*) |
| Summary | Returns a new array with those members of $array for which $function returns true. |
| Examples | The following query returns the array [0, 1, 3]: |
|         | array:filter( |
|         |   array ( 0, 1, -2, 3, -4 ), |
|         |   function($i) ( $i > 0 ) |
|         | ) |

#### array:fold-left

| Signatures | array:fold-left($array as array(*), $zero as item()*, $function as function(item()*, item()) as item()) as item() |
| Summary | Evaluates the supplied $function cumulatively on successive members of the supplied $array from left to right and using $zero as first argument. |
| Examples | The following query returns 55 (the sum of the integers 1 to 10): |
|         | array:fold-left( |
|         |   array ( 1 to 10 ), |
|         |   0, |
|         |   function($a, $b) { $a + $b } |
|         | ) |
array:fold-right

**Signatures**

array:fold-right($array as array(*), $zero as item(*), $function as function(item(*), item(*)) as item(*)) as item(*)

**Summary**

Evaluates the supplied $function cumulatively on successive members of the supplied $array from right to left and using $zero as first argument.

**Examples**

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

```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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOAY0001</td>
<td>The specified index extends beyonds the bounds of an array.</td>
</tr>
<tr>
<td>FOAY0002</td>
<td>The specified length is less than zero.</td>
</tr>
</tbody>
</table>
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('11010001111010101')) yields 0dU=. string(xs:hexBinary(bin:bin('1000111010101'))) 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**

**Signatures**

`bin:to-octets($in as xs:base64Binary) as xs:integer*`

**Summary**

Returns binary data as a sequence of octets. If `$in` is a zero length binary data then the empty sequence is returned. Octets are returned as integers from 0 to 255.

**bin:from-octets**

**Signatures**

`bin:from-octets($in as xs:integer*) as xs:base64Binary`

**Summary**

Converts a sequence of octets into binary data. Octets are integers from 0 to 255. If the value of `$in` is the empty sequence, the function returns zero-sized binary data.

**Errors**

`octet-out-of-range`: one of the octets lies outside the range 0 - 255.

**Basic Operations**

**bin:length**

**Signatures**

`bin:length($in as xs:base64Binary) as xs:integer`

**Summary**

Returns the size of binary data in octets.

**bin:part**

**Signatures**

`bin:part($in as xs:base64Binary?, $offset as xs:integer) as xs:base64Binary?`

`bin:part($in as xs:base64Binary?, $offset as xs:integer, $size as xs:integer) as xs:base64Binary?`

**Summary**

Returns a section of binary data starting at the `$offset` octet. If `$size` is specified, the size of the returned binary data is `$size` octets. If `$size` is absent, all remaining data from `$offset` is returned. The `$offset` is zero based. If the value of `$in` is the empty sequence, the function returns an empty sequence.

**Errors**

`negative-size`: the specified size is negative. `index-out-of-range`: the specified offset + size is out of range.

**Examples**

Test whether binary data starts with binary content consistent with a PDF file:

```
bin:part($data, 0, 4) eq bin:hex("25504446").
```

**bin:join**

**Signatures**

`bin:join($in as xs:base64Binary*) as xs:base64Binary`

**Summary**

Returns an `xs:base64Binary` created by concatenating the items in the sequence `$in`, in order. If the value of `$in` is the empty sequence, the function returns a binary item containing no data bytes.

**bin:insert-before**

**Signatures**

`bin:insert-before($in as xs:base64Binary?, $offset as xs:integer, $extra as xs:base64Binary?) as xs:base64Binary?`

**Summary**

Returns binary data consisting sequentially of the data from `$in` up to and including the `$offset` - 1 octet, followed by all the data from `$extra`, and then the remaining data from `$in`. The `$offset` is zero based. If the value of `$in` is the empty sequence, the function returns an empty sequence.

**Errors**

`index-out-of-range`: the specified offset is out of range.
**bin:pad-left**

**Signatures**
\[
\text{bin:pad-left}(\$in \text{ as } \text{xs:base64Binary}?, \$size \text{ as } \text{xs:integer}) \text{ as } \text{xs:base64Binary}?
\]

**Summary**
Returns an \text{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**
\[
\text{bin:pad-right}(\$in \text{ as } \text{xs:base64Binary}?, \$size \text{ as } \text{xs:integer}) \text{ as } \text{xs:base64Binary}?
\]

**Summary**
Returns an \text{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**
\[
\text{bin:find}(\$in \text{ as } \text{xs:base64Binary}?, \$offset \text{ as } \text{xs:integer}, \$search \text{ as } \text{xs:base64Binary}) \text{ as } \text{xs:integer}?
\]

**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**
\[
\text{bin:decode-string}(\$in \text{ as } \text{xs:base64Binary}?, \$encoding \text{ as } \text{xs:string}) \text{ as } \text{xs:string}?
\]

**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**
\[
\text{bin:encode-string}(\$in \text{ as } \text{xs:string}?, \$encoding \text{ as } \text{xs:string}) \text{ as } \text{xs:base64Binary}?
\]
### Binary Module

**Summary**
Encodes a string into binary data using a given $encoding$. If the value of $in$ is the empty sequence, the function returns an empty sequence.

**Errors**
- unknown-encoding: the specified encoding is unknown.
- conversion-error: an error or malformed input occurred during encoding the string.

## Packing and Unpacking of Numeric Values

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

### bin:pack-double

**Signatures**
- `bin:pack-double($in as xs:double) as xs:base64Binary`
- `bin:pack-double($in as xs:double, $octet-order as xs:string) as xs:base64Binary`

**Summary**
Returns the 8-octet binary representation of a double value. Most-significant-octet-first number representation is assumed unless the $octet-order$ parameter is specified.

**Errors**
- unknown-significance-order: the specified octet order is unknown.

### bin:pack-float

**Signatures**
- `bin:pack-float($in as xs:float) as xs:base64Binary`
- `bin:pack-float($in as xs:float, $octet-order as xs:string) as xs:base64Binary`

**Summary**
Returns the 4-octet binary representation of a float value. Most-significant-octet-first number representation is assumed unless the $octet-order$ parameter is specified.

**Errors**
- unknown-significance-order: the specified octet order is unknown.

### bin:pack-integer

**Signatures**
- `bin:pack-integer($in as xs:integer, $size as xs:integer) as xs:base64Binary`
- `bin:pack-integer($in as xs:integer, $size as xs:integer, $octet-order as xs:string) as xs:base64Binary`

**Summary**
Returns the two's-complement binary representation of an integer value treated as $size$ octets long. Any 'excess' high-order bits are discarded. Most-significant-octet-first number representation is assumed unless the $octet-order$ parameter is specified. Specifying a $size$ of zero yields an empty binary data.

**Errors**
- unknown-significance-order: the specified octet order is unknown.
- negative-size: the specified size is negative.

### bin:unpack-double

**Signatures**
- `bin:unpack-double($in as xs:base64Binary, $offset as xs:integer) as xs:double`
- `bin:unpack-double($in as xs:base64Binary, $offset as xs:integer, $octet-order as xs:string) as xs:double`

**Summary**
Extracts the double value stored at the particular offset in binary data. Most-significant-octet-first number representation is assumed unless the $octet-order$ parameter is specified. The $offset$ is zero based.

**Errors**
- index-out-of-range: the specified offset is out of range.
- unknown-significance-order: the specified octet order is unknown.
**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?
```
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', 'admin') 
let $docs := client:query($client-id, 'db:open("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 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', 'admin') ! 
client:execute(., 'create database TEST')

client:info

Signatures  
client:info($id as xs:anyURI) as xs:string
**Client Module**

### Summary
This function returns an information string, created by the last call of `client:execute`. `$id` specifies the session id.

### client:query

#### Signatures

```
client:query($id as xs:anyURI, $query as xs:string) as item()*
client:query($id as xs:anyURI, $query as xs:string, $bindings as map(*)?) as item()*
```

#### Summary
Evaluates a query and returns the result as sequence. The parameter `$id` contains the session id returned by `client:connect`, and `$query` represents the query string, which will be evaluated by the server. Variables and the context item can be declared via `$bindings`. The specified keys must be QNames or strings:

- If a key is a QName, it will be directly adopted as variable name.
- If a key is a string, it may be prefixed with a dollar sign. A namespace can be specified using the Clark Notation. If the specified string is empty, the value will be bound to the context item.

#### Errors

- `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:

```
let $c := client:connect('localhost', 1984, 'admin', 'admin')
return client:query($c, "declare variable $n external; $n * 2", map ('n': 123 ))
```

The following query performs a query on a first server, the results of which are passed on to a second server:

```
let $c1 := client:connect('baseX1.server.org', 8080, 'jack', 'C0S19tt2X')
let $c2 := client:connect('baseX2.server.org', 8080, 'john', '465WFHe26')
for $it in client:query($c1, '1 to 10')
return client:query($c2, $it || '* 2')
```

### client:close

#### Signatures

```
client:close($id as xs:anyURI) as empty-sequence()
```

#### Summary
This function closes a client session. `$id` specifies the session id. 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>command</td>
<td>An error occurred while executing a command.</td>
</tr>
<tr>
<td>connect</td>
<td>An error occurred while creating a new session (possible reasons: server not available, access denied).</td>
</tr>
<tr>
<td>error</td>
<td>An I/O error occurred while transferring data from or to the server.</td>
</tr>
<tr>
<td>function</td>
<td>Function items (including maps and arrays) cannot be returned.</td>
</tr>
<tr>
<td>id</td>
<td>The id with the specified session is unknown, or has already been closed.</td>
</tr>
</tbody>
</table>
query | An error occurred while evaluating a query. Will only be raised if the XQuery error cannot be extracted from the returned error string.

**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 client:query.

Version 7.5

• Added: client:info

The module was introduced with Version 7.3.
Chapter 39. Conversion Module

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 SGVsbG9Xb3JsZAA==.

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.

### Examples
- `string(convert:string-to-hex('HelloWorld'))` yields `48656C6C6F576F726C64`.

## Binary Data

### `convert:integers-to-base64`

**Signatures**

convert:integers-to-base64($integers as xs:integer*) as xs:base64Binary

**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 particularly cheap, as items of binary type are internally represented as byte arrays.

### `convert:integers-to-hex`

**Signatures**

convert:integers-to-hex($integers as xs:integer*) as xs:hexBinary

**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 particularly cheap, as items of binary type are internally represented as byte arrays.

### `convert:binary-to-integers`

**Signatures**

convert:binary-to-integers($binary as xs:anyAtomicType) as xs:integer*

**Summary**

Returns the specified $binary (xs:base64Binary, xs:hexBinary) as a sequence of unsigned integers (octets).

**Examples**

- `convert:binary-to-integers(xs:hexBinary('FF'))` yields 255.

### `convert:binary-to-bytes`

**Signatures**

convert:binary-to-bytes($binary as xs:anyAtomicType) as xs:byte*

**Summary**

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.

**Examples**

- `convert:binary-to-bytes(xs:base64Binary('QmFzZVggaXMgY29vbA=='))` yields the sequence `{66, 97, 115, 101, 88, 32, 105, 115, 32, 99, 111, 111, 108}`.

## Numbers

### `convert:integer-to-base`

**Signatures**

convert:integer-to-base($number as xs:integer, $base as xs:integer) as xs:string
Conversion Module

Summary
Converting $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 milliseconds 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
Conversion Module

Summary
Converts the specified number of $\text{milliseconds}$ to an item of type $\text{xs:dayTimeDuration}$.

Examples
• convert:integer-to-dayTime(1234) yields PT1.234S.

convert:dayTime-to-integer

Signatures
convert:dayTime-to-integer($\text{dayTime}$ as $\text{xs:dayTimeDuration}$) as $\text{xs:integer}$

Summary
Converts the specified $\text{dayTime}$ duration to milliseconds represented by an integer.

Examples
• convert:dayTime-to-integer($\text{xs:dayTimeDuration('PT1S')}\text{)}$ yields 1000.

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>
<tr>
<td>encoding</td>
<td>The specified encoding is invalid or not supported.</td>
</tr>
<tr>
<td>integer</td>
<td>The specified digit is not valid for the given range.</td>
</tr>
<tr>
<td>string</td>
<td>The input is an invalid XML string, or the wrong encoding has been specified.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.0
• Added: convert:binary-to-integers.

• Updated: convert:integers-to-base64, convert:integers-to-hex: Renamed from convert:bytes-to-base64; argument type relaxed from $\text{xs:byte}$ to $\text{xs:integer}$.

• Updated: error codes updated; errors now use the module namespace

Version 8.5
• Updated: convert:binary-to-string: $\text{fallback}$ argument added.

Version 7.5
• Added: convert:integer-to-dateTime, convert:dateTime-to-integer, convert:integer-to-dayTime, convert:dayTime-to-integer.

The module was introduced with Version 7.3. Some of the functions have been adopted from the obsolete Utility Module.
Chapter 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($message as xs:string, $key as xs:anyAtomicType) as xs:base64Binary, $algorithm as xs:string crypto:hmac($message as xs:string, $key as xs:anyAtomicType) as xs:base64Binary, $algorithm as xs:string, $encoding as xs:string

Summary

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

Errors

CX0013: the specified hashing algorithm is not supported. CX0014: the specified encoding method is not supported. CX0019: the specified secret key is invalid.

Example

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

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

Result:

34D1E3818B347252A75A4F6D747B21C2

Encryption & Decryption

The encryption and decryption functions underlie several limitations:

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

**Signatures**
crypto:encrypt($input as xs:string, $encryption as xs:string, $key as xs:string, $algorithm as xs:string) as xs:string

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

**Errors**
CX0016: padding problems arise. CX0017: padding is incorrect. CX0018: the encryption type is not supported. CX0019: the secret key is invalid. CX0020: the block size is incorrect. CX0021: the specified encryption algorithm is not supported.

**Example**
Encrypts input data. Query:

crypto:encrypt('message', 'symmetric','keykeyke', 'DES')

**crypto:decrypt**

**Signatures**
crypto:decrypt($input as xs:string, $type as xs:string, $key as xs:string, $algorithm as xs:string) as xs:string

**Summary**
Decrypts the encrypted $input. $type must be symmetric. An option for asymmetric encryption will most likely be added with another version of BaseX. **Default is symmetric**. $key is the secret key which is used for both encryption and decryption of input data. Its length is fixed and depends on the chosen algorithm: 8 bytes for DES, 16 bytes for AES. $algorithm must either be DES or AES. Other algorithms are not supported so far, but, of course, can be added on demand. **Default is DES**.

**Errors**
CX0016: padding problems arise. CX0017: padding is incorrect. CX0018: the encryption type is not supported. CX0019: the secret key is invalid. CX0020: the block size is incorrect. CX0021: the specified encryption algorithm is not supported.

**Example**
Decrypts input data and returns the original string. Query:

```xml
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**

```xml
<Signature>
    <SignedInfo>
        <CanonicalizationMethod/>
        <SignatureMethod/>
    </SignedInfo>
    <Reference>
        <Transforms/>
        <DigestMethod/>
        <DigestValue/>
    </Reference>
</Signature>
```
Cryptographic Module

- **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>
```

```xml
crypto:generate-signature
```

**Signatures**

```xml
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, $certificate as node(), $ext as item()) as node()
```

**Summary**

$canonicalization must either be inclusive-with-comments, inclusive, exclusive-with-comments or exclusive. Default is inclusive-with-comments. $digest must be one of the following: SHA1, SHA256 or SHA512. Default is SHA1. $signature must either be RSA_SHA1 or DSA_SHA1. Default is RSA_SHA1. $prefix may be empty and prefixes the Signature element accordingly. $type is the signature type. It must...
either be enveloped or enveloping (detached signatures are not supported so far). **Default is enveloped.** $\$xpath$ is an arbitrary XPath expression which specifies a subset of the document that is to be signed. $\$certificate$ is the digital certificate used to sign the input document. $\$ext$ may either be an $\$xpath$ expression or a $\$certificate$.

**Errors**

- CX0001: the canonicalization algorithm is not supported.
- CX0002: the digest algorithm is not supported.
- CX0003: the signature algorithm is not supported.
- CX0004: the $\$xpath$-expression is invalid.
- CX0005: the root name of $\$digital-certificate$ is not 'digital-certificate'.
- CX0007: the key store is null.
- CX0012: the key cannot be found in the specified key store.
- CX0023: the certificate alias is invalid.
- CX0024: an invalid algorithm is specified.
- CX0025: an exception occurs while the signing the document.
- CX0026: an exception occurs during key store initialization.
- CX0027: an IO exception occurs.
- CX0028: the signature type is not supported.

**Example**

**Generates an XML Signature.** Query:

```xml
crypto:generate-signature(<a/>, '', '', '', '', '')
```

**Result:**

```xml
<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>9hvH4qztn1YgYfJDRLnEMPJdoaY==</DigestValue>
    </Reference>
    <SignatureValue>Pn/Jr44WBcdARff2UVYEiwyYW1563XdqnU87nusAIaHgzd+U3SrjVjhpFPLDe0DJfxfVtzylFaznTYE P3deoFmyA==</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>rtvpFSbCIE2BJeP1VYLIRijX10R7ESr2+D+J0VKn7AM7VzbcbRDPegRbjSkEz1HWC/N067tjb3qH 4/4PPT9bQ==</Modulus>
          <Exponent>AQAB</Exponent>
        </RSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</a>
```

**crypto:validate-signature**

**Signatures**

crypto:validate-signature($\$input-doc$ as node()) as xs:boolean

**Summary**

Checks if the given node contains a Signature element and whether the signature is valid. In this case true is returned. If the signature is invalid the function returns false.

**Errors**

- CX0015: the signature element cannot be found.
- CX9994: an unspecified problem occurs during validation.
- CX9996: an IO exception occurs during validation.

**Example**

**Validates an XML Signature.** Query:
let $sig := crypto:generate-signature(<a/>, '', '', '', '', '')
return crypto:validate-signature($sig)

Result:
true

Errors

<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>CX0023</td>
<td>An invalid certificate alias is specified. Added to the official specification.</td>
</tr>
<tr>
<td>CX0024</td>
<td>The algorithm is invalid. Added to the official specification.</td>
</tr>
<tr>
<td>CX0025</td>
<td>Signature cannot be processed. Added to the official specification.</td>
</tr>
<tr>
<td>CX0026</td>
<td>Keystore cannot be processed. Added to the official specification.</td>
</tr>
<tr>
<td>CX0027</td>
<td>An I/O Exception occurred. Added to the official specification.</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 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, and the <entry> elements are replaced with the field names:
  • Empty names are represented by a single underscore (_), and characters that are not valid in element names are replaced with underscores or (when invalid as first character of an element name) prefixed with an underscore.
  • If the lax option is set to false, invalid characters will be rewritten to an underscore and the character’s four-digit Unicode, and underscores will be represented as two underscores (__). The resulting element names may be less readable, but can always be converted back to the original field names.
• If format is set to attributes, field names will be stored in name attributes.

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 indicates what are the preferred options for data that is to be imported, or has been exported from Excel.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Allowed</th>
<th>Default</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>semicolon</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></td>
</tr>
<tr>
<td>format</td>
<td>Specifies the format of the XML data:</td>
<td>direct, attributes, xquery</td>
<td>direct</td>
<td>attributes, xquery</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>no</td>
</tr>
<tr>
<td>quotes</td>
<td>Specifies how quotes are parsed:</td>
<td>yes, no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>backslashes</td>
<td>Specifies how quotes and other characters are escaped:</td>
<td>yes, no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

205
Functions

**csv:parse**

*Updated with Version 9.1:* support for empty sequence.

<table>
<thead>
<tr>
<th>Signatures</th>
<th>csv:parse($string as xs:string?) as item()? csv:parse($string as xs:string?, $options as map(*)?) as item()?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Converts the CSV $string to an XQuery value. The $options argument can be used to control the way the input is converted.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>parse: the input cannot be parsed.</td>
</tr>
</tbody>
</table>

**csv:serialize**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>csv:serialize($input as item()?) as xs:string csv:serialize($input as item()?, $options as map(*)?) as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>• The parameter method needs to be set to csv, and</td>
</tr>
<tr>
<td></td>
<td>• the options presented in this article need to be assigned to the csv parameter.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>serialize: the input cannot be serialized.</td>
</tr>
</tbody>
</table>

Examples

**Example 1:** Converts CSV data to XML, interpreting the first row as table header:

**Input** addressbook.csv:

<table>
<thead>
<tr>
<th>Name,First Name,Address,City</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huber,Sepp,Hauptstraße 13,93547 Hintertupfing</td>
</tr>
</tbody>
</table>

**Query:**

```xquery
let $text := file:read-text('addressbook.csv')
return csv:parse($text, map { 'header': true() })
```

**Result:**

```xml
<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:**

```xquery
let $options := map { 'lax': false() }
```

206
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:**

```xml
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: ' || $name || ', ' || string-join(distinct-values($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 9.1**

- 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

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for processing databases from within XQuery. Existing databases can be opened and listed, its contents can be directly accessed, documents can be added to and removed, etc.

Conventions

All functions 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

Database nodes are XML nodes which are either stored in a persistent database, or which are a node of a main-memory database representation. XML fragments can be converted to a main-memory database by e.g. applying the update or transform expression on a node:

\[
\text{db:node-id(element hello { 'world' } update { })}
\]

General Functions

**db:system**

**Signatures**

\[
\text{db:system()} \text{ as element(system)}
\]

**Summary**

Returns general information on the database system the current values of all global and local Options. The INFO command returns similar output.

**db:option**

**Signatures**

\[
\text{db:option($name as xs:string) as xs:string}
\]

**Summary**

Returns the current value (string, integer, boolean, map) of a global or local Option with the specified $name. The GET command works similar.

**Examples**

- \[
\text{db:option('dbpath')} \text{ returns the database path string.}
\]

- \[
\text{db:option('serializer')} \text{ returns a map with the current serialization parameters.}
\]

- \[
\text{declare option db:chop 'true'; db:option('chop')} \text{ returns true (irrespective of the global value).}
\]

**db:info**

**Signatures**

\[
\text{db:info($db as xs:string) as element(database)}
\]

**Summary**

Returns meta information on the database $db. The output is similar to the INFO DB command.

**Errors**

open: the addressed database does not exist or could not be opened.

**db:property**

**Signatures**

\[
\text{db:property($db as xs:string, $name as xs:string) as xs:anyAtomicType}
\]

**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.

### db:list

**Signatures**

- `db:list() as xs:string*`
- `db:list($db as xs:string) as xs:string*`
- `db:list($db as xs:string, $path as xs:string) as xs:string*`

**Summary**

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.

### db:list-details

**Signatures**

- `db:list-details() as element(database)*`
- `db:list-details($db as xs:string) as element(resource)*`
- `db:list-details($db as xs:string, $path as xs:string) as element(resource)*`

**Summary**

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 $db is specified, an element for each documents 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 $path argument.

**Errors** | open: the addressed database does not exist or could not be opened.
--- | ---
**Examples** | • `db:list-details("shop")` returns the names plus additional info on all resources of a database named shop.

### db:backups

**Signatures**

- `db:backups() as element(backup)*`
- `db:backups($db as xs:string) as element(backup)*`

**Summary**

Returns an element sequence containing all available database backups. If a database $db is specified, the sequence will be restricted to the backups matching this database.
Examples

- `db:backups("factbook")` returns all backups that have been made from the `factbook` database.

**Read Operations**

### `db:open`

**Signatures**
```
db:open($db as xs:string) as document-node()*
db:open($db as xs:string, $path as xs:string) as document-node()*
```

**Summary**
Opens the database `$db` and returns all document nodes. The document nodes to be returned can be filtered with the `$path` argument.

**Errors**
- `open`: the addressed database does not exist or could not be opened.

**Examples**
- `db:open("docs")` returns all documents from the database named `docs`.
- `db:open("db", "one")` returns all documents from the database named `db` located in the path `one`.
- `for $i in 1 to 3 return db:open("db" || $i)//item` returns all item elements from the databases `db1`, `db2` and `db3`.

### `db:open-pre`

**Signatures**
```
db:open-pre($db as xs:string, $pre as xs:integer) as node()
```

**Summary**
Opens the database `$db` and returns the node with the specified `$pre` value. The `PRE value` provides very fast access to an existing database node, but it will change whenever a node with a smaller `pre` value 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:open-pre("docs", 0)` returns the first database node from the database named `docs`.

### `db:open-id`

**Signatures**
```
db:open-id($db as xs:string, $id as xs:integer) as node()
```

**Summary**
Opens the database `$db` and returns the node with the specified `$id` value. Each database node has a `persistent ID value`. Access to the node id can be sped up by turning on the `UPDINDEX` option.

**Errors**
- `open`: the addressed database does not exist or could not be opened.
- `range`: the specified id value does not exist in the database.

### `db:node-pre`

**Signatures**
```
db:node-pre($nodes as node()*) as xs:integer*
```

**Summary**
Returns the `pre` values of the nodes supplied by `$nodes`, which must all be database nodes. The `PRE value` provides very fast access to an existing database node, but it will change whenever a node with a smaller `pre` value 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*
```
### database_module

#### Summary
Returns the id values of the nodes supplied by $nodes, which must all be database nodes. Each database node has a persistent ID value. Access to the node id can be sped up by turning on the UPDINDEX option.

#### Errors
- node: $nodes contains a node which is not stored in a database.

#### db:retrieve

**Signatures**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db:retrieve($db as xs:string, $path as xs:string) as xs:base64Binary</td>
<td></td>
</tr>
</tbody>
</table>

**Summary**
Returns a binary resource addressed by the database $db and $path as streamable xs:base64Binary.

**Errors**
- open: the addressed database does not exist or could not be opened.
- mainmem: the database is not persistent (stored on disk).

**Examples**

- db:retrieve("DB", "music/01.mp3") returns the specified audio file as raw data.
- stream:materialize(db:retrieve("DB", "music/01.mp3")) materializes the streamable result in main-memory before returning it.
- convert:binary-to-string(db:retrieve("DB", "info.txt"), 'UTF-8') converts a binary database resource as UTF-8 text and returns a string.

#### db:export

**Signatures**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db:export($db as xs:string, $path as xs:string) as empty-sequence()</td>
<td></td>
</tr>
<tr>
<td>db:export($db as xs:string, $path as xs:string, $params as item()) as empty-sequence()</td>
<td></td>
</tr>
</tbody>
</table>

**Summary**
Exports the specified database $db to the specified file $path. Existing files will be overwritten. The $params argument contains serialization parameters (see Serialization for more details), which can either be specified

- as children of an `<output:serialization-parameters/>` element, as defined for the fn:serialize() function; e.g.:

  ```xml
  <output:serialization-parameters>
  <output:method value='xml'/>
  <output:cdata-section-elements value="div"/>
  ...
  </output:serialization-parameters>
  ```

- as map, which contains all key/value pairs:

  ```xml
  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:

```xml
db:export("DB", "/home/john/xml/texts", map { 'method': 'text' })
```

The following query can be used to export parts of the database:

```xml
let $target := '/home/john/xml/target'
for $doc in db:open('DB', 'collection')
let $path := $target || db:path($doc)
return {
  file:create-dir(file:parent($path)),
  ```
### Value Indexes

**db:text**

*Updated with Version 9.1:* Support for multiple string values.

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>db:text($db as xs:string, $strings as xs:string*) as text()*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td><code>open</code>: the addressed database does not exist or could not be opened. <code>no-index</code>: the index is not available.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td><code>• db:text(&quot;DB&quot;, &quot;QUERY&quot;)/../</code> returns the parents of all text nodes of the database DB that match the string QUERY.</td>
</tr>
</tbody>
</table>

**db:text-range**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>db:text-range($db as xs:string, $min as xs:string, $max as xs:string) as text()*</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Returns all text nodes of the database $db that are located in between the $min and $max strings and that are stored in the text index.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td><code>open</code>: the addressed database does not exist or could not be opened. <code>no-index</code>: the index is not available.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td><code>• db:text-range(&quot;DB&quot;, &quot;2000&quot;, &quot;2001&quot;)</code> returns all text nodes of the database DB that are found in between 2000 and 2001.</td>
</tr>
</tbody>
</table>

**db:attribute**

*Updated with Version 9.1:* Support for multiple string values.

| Signatures       | `db:attribute($db as xs:string, $strings as xs:string*) as attribute()*` | `db:attribute($db as xs:string, $strings as xs:strings*, $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**

*Updated with Version 9.1: Support for multiple string values.*

**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**

*Important note:* All functions in this section are *updating functions*: they will not be immediately executed, but queued on the Pending Update List, which will be processed after the actual query has been evaluated. This means that the order in which the functions are specified in the query does usually not reflect the order in which the code will be evaluated.

**db:create**

**Signatures**

- `db:create($db as xs:string) as empty-sequence()`
- `db:create($db as xs:string, $inputs as item()* as empty-sequence())`
- `db:create($db as xs:string, $inputs as item(), $paths as xs:string*) as empty-sequence()`
- `db:create($db as xs:string, $inputs as item(), $paths as xs:string*, $options as map()?) 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:drop**

**Signatures**

```
db:drop($db as xs:string) as empty-sequence()
```

**Summary**

Drops the database $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**

- `db:drop("DB")` drops the database DB.

**db:add**

**Signatures**

```
db:add($db as xs:string, $input as item()) as empty-sequence()
db:add($db as xs:string, $input as item(), $path as xs:string) as empty-sequence()
db:add($db as xs:string, $input as item(), $path as xs:string, $options as map(*)?) 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:replace` instead.
- See `db:create` for more details on the input argument.
- 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.

**Examples**

- `db:add("DB", "/home/dir/doc.xml")` adds the file `/home/dir/doc.xml` to the database DB.
- `db:add("DB", <a/>, "doc.xml")` adds a document node to the database DB under the name `doc.xml`.
- `db:add("DB", "/home/dir/", "docs/dir", map { 'addcache': true() })` adds all documents in `/home/dir` to the database DB under the path `docs/dir`. In order to reduce memory consumption, the files will be cached before being added to the database.
### Database Module

<table>
<thead>
<tr>
<th><strong>Summary</strong></th>
<th>Deletes resource(s), specified by $path, from the database $db.</th>
</tr>
</thead>
</table>

| **Errors** | open: the addressed database does not exist or could not be opened.path: the specified path is invalid. |

| **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>
</table>

| **Summary** | Creates a copy of the database $db, which will be called $name. |

| **Errors** | 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. |

### $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>
</table>

| **Summary** | Renames the database $db to $name. |

| **Errors** | 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. |

### $db:create-backup

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>$db:create-backup($db as xs:string) as empty-sequence()</th>
</tr>
</thead>
</table>

| **Summary** | Creates a backup of the database $db. |

| **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** | • $db:create-backup("DB") creates a backup of the database DB. |

### $db:drop-backup

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>$db:drop-backup($name as xs:string) as empty-sequence()</th>
</tr>
</thead>
</table>

| **Summary** | Drops all backups of the database with the specified $name. If the given $name points to a specific backup file, only this specific backup file is deleted. |

| **Errors** | backup: No backup file found.name: invalid database name.conflict: the same database was addressed more than once. |

| **Examples** | • $db:drop-backup("DB") drops all backups of the database DB.  
• $db:drop-backup("DB-2014-03-13-17-36-44") drops the specific backup file DB-2014-03-13-17-36-44.zip of the database DB. |

### $db:restore

<table>
<thead>
<tr>
<th><strong>Signatures</strong></th>
<th>$db:restore($name as xs:string) as empty-sequence()</th>
</tr>
</thead>
</table>

| **Summary** | Restores the database with the specified $name. The $name may include the timestamp of the backup file. |

| **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. |
Database Module

**Examples**

- `db:restore("DB")` restores the database DB.
- `db:restore("DB-2014-03-13-18-05-45")` restores the database DB from the backup file with the given timestamp.

**db:optimize**

**Signatures**

```
db:optimize($db as xs:string) as empty-sequence()
db:optimize($db as xs:string, $all as xs:boolean) as empty-sequence()
db:optimize($db as xs:string, $all as xs:boolean, $options as map(*)) as empty-sequence()
```

**Summary**

Optimizes the meta data and indexes of the database `$db`. If `$all` is true, the complete database will be rebuilt. The `$options` argument can be used to control indexing. The syntax is identical to the `db:create` function. Allowed options are all indexing and full-text options. `UPDINDEX` is only supported if `$all` is true.

**Errors**

- `open`: the addressed database does not exist or could not be opened.

**Examples**

- `db:optimize("DB")` optimizes the database structures of the database DB.
- `db:optimize("DB", true(), map { 'ftindex': true() })` optimizes all database structures of the database DB and creates a full-text index.

**db:rename**

**Signatures**

```
db:rename($db as xs:string, $source as xs:string, $target as xs:string) as empty-sequence()
```

**Summary**

Moves all resources(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.

**Errors**

- `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.

**Examples**

- `db:rename("DB", "docs/dir/doc.xml", "docs/dir/newdoc.xml")` renames the resource `docs/dir/doc.xml` to `docs/dir/newdoc.xml` in the database DB.
- `db:rename("DB", "docs/dir", "docs/newdir")` moves all resources in the database DB from `docs/dir` to `{Code|docs/newdir}`.

**db:replace**

**Signatures**

```
db:replace($db as xs:string, $path as xs:string, $input as item()) as empty-sequence()
db:replace($db as xs:string, $path as xs:string, $input as item(), $options as 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:

- See `db:create` for more details on the input argument.
- 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
  - For historical reasons, the order of the 2nd and 3rd argument is different to `db:add` and `db:create`

**Errors**

- `open`: the addressed database does not exist or could not be opened.
- `target`: the path points to a directory.
Examples

- `db:replace("DB", "docs/dir/doc.xml", "/home/dir/doc.xml")` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with the content of the file `/home/dir/doc.xml`.

- `db:replace("DB", "docs/dir/doc.xml", "<a/>")` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with `<a/>`.

- `db:replace("DB", "docs/dir/doc.xml", document { <a/> })` replaces the content of the document `docs/dir/doc.xml` in the database `DB` with the specified document node.

The following query can be used to import files from a directory to a database:

```xml
let $source := '/home/john/xml/source'
for $file in file:list($source, true())
let $path := $source || $file
where not(file:is-dir($path))
return db:replace('db', $file, doc($path))
```

**db:store**

**Signatures**

`db:store($db as xs:string, $path as xs:string, $input as item()) as empty-sequence()`

**Summary**

Replaces a binary resource specified by `$input` in the database `$db` and the location specified by `$path`, or adds it as new resource.

**Errors**

- `open`: the addressed database does not exist or could not be opened.
- `mainmem`: the database is not persistent (stored on disk).

**Examples**

- `db:store("DB", "video/sample.mov", file:read-binary('video.mov'))` stores the addressed video file at the specified location.

- With the following query, you can copy full directories:

  ```xml
  let $db := 'db'
  let $src-path := 'src/'
  let $trg-path := 'trg/'
  for $src in db:list($db, $src-path)
  where db:is-raw($db, $src)
  let $trg := $trg-path || substring-after($src, $src-path)
  return db:store($db, $trg, db:retrieve($db, $src))
  ```

**db:flush**

**Signatures**

`db:flush($db as xs:string) as empty-sequence()`

**Summary**

Explicitly flushes the buffers of the database `$db`. This command is only useful if `AUTOFLUSH` has been set to `false`.

**Errors**

- `open`: the addressed database does not exist or could not be opened.

**Helper Functions**

**db:name**

**Signatures**

`db:name($node as node()) as xs:string`

**Summary**

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

**Errors**

- `node`: `$node` 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 $db or the resource specified by $path exists. false is returned if a database directory has been addressed.

**Examples**  
• db:exists("DB") returns true if the database DB exists.
• db:exists("DB", "resource") returns true if resource is an XML document or a raw file.

### db:is-raw

**Signatures**  
db:is-raw($db as xs:string, $path as xs:string) as xs:boolean  

**Summary**  
Checks if the specified resource in the database $db and the path $path exists, and if it is a binary resource.

**Errors**  
open: the addressed database does not exist or could not be opened.

**Examples**  
• db:is-raw("DB", "music/01.mp3") returns true.

### db:is-xml

**Signatures**  
db:is-xml($db as xs:string, $path as xs:string) as xs:boolean  

**Errors**  
open: the addressed database does not exist or could not be opened.

**Summary**  
Checks if the specified resource in the database $db and the path $path exists, and if it is an XML document.

**Examples**  
• db:is-xml("DB", "dir/doc.xml") returns true.

### db:content-type

**Signatures**  
db:content-type($db as xs:string, $path as xs:string) as xs:string  

**Summary**  
Retrieves the content type of a resource in the database $db and the path $path. The file extension is used to recognize the content-type of a resource stored in the database. Content-type application/xml will be returned for any XML document stored in the database, regardless of its file name extension.

**Errors**  
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>
</table>
Database Module

<table>
<thead>
<tr>
<th>Arg</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>path</td>
<td>The specified database path is invalid.</td>
</tr>
<tr>
<td>property</td>
<td>A database property is unknown.</td>
</tr>
<tr>
<td>range</td>
<td>The addressed database id or pre value is out of range.</td>
</tr>
<tr>
<td>target</td>
<td>Path points to an invalid target.</td>
</tr>
</tbody>
</table>

**Changelog**

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`. 

220
Version 7.8.2
- Added: db:alter, db:copy, db:create-backup, db:drop-backup, db:restore

Version 7.8
- Removed: db:fulltext (use ft:search instead)

Version 7.7
- Added: db:export, db:name, db:path
- Updated: $options argument added to db:create and db:optimize.
- Updated: the functions no longer accept Database Nodes as reference. Instead, the name of a database must now be specified.

Version 7.6
- Updated: db:create: allow more than one input and path.

Version 7.5
- Updated: db:add: input nodes will be automatically converted to document nodes
- Added: db:backups
- Added: db:create
- Added: db:drop

Version 7.3
- Added: db:flush

Version 7.2.1
- Added: db:text-range, db:attribute-range, db:output

Version 7.1
- Added: db:list-details, db:content-type
- Updated: db:info, db:system, db:retrieve

Version 7.0
- Added: db:retrieve, db:store, db:exists, db:is-raw, db:is-xml
- Updated: db:list, db:open, db:add
Chapter 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 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
Fetched 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

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 Module

**fetch:xml**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>fetch:xml($uri as xs:string) as document-node() fetch:xml($uri as xs:string, $options as map(*)?) as document-node()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Fetches the resource referred to by the given $uri and returns it as XML document node. In contrast to fn:doc, each function call returns a different document node. As a consequence, document instances created by this function will not be kept in memory until the end of query evaluation. The $options argument can be used to change the parsing behavior. Allowed options are all parsing and XML parsing options in lower case.</td>
</tr>
<tr>
<td><strong>Errors</strong></td>
<td>open: the URI could not be resolved, or the resource could not be retrieved.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>• Retrieve an XML representation of the English Wikipedia main HTML page, chop all whitespace nodes:</td>
</tr>
</tbody>
</table>
|                     | ```xml
defetch:xml("http://en.wikipedia.org", map ( 'chop': true() ))
```
|                     | • Return a document located in the current base directory: |
|                     | ```xml
defetch:xml(file:base-dir() || "example.xml")
```
|                     | • Return a web page as XML, preserve namespaces: |
|                     | ```xml
defetch:xml(
  'http://basex.org/",
  map (  
    'parser': 'html',
    'htmlparser': map ( 'nons': false() )
  )
)
```

**fetch:xml-binary**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>fetch:xml-binary($data as xs:base64Binary) as document-node() fetch:xml-binary($data as xs:base64Binary, $options as map(*)?) as document-node()</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Parses binary $data and returns it as XML document node. In contrast to fn:parse-xml, which expects an XQuery string, the input of this function can be arbitrarily encoded. The encoding will be derived from the XML declaration or (in case of UTF16 or UTF32) 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.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>• Retrieves file input as binary data and parses it as XML:</td>
</tr>
</tbody>
</table>
|                     | ```xml
defetch:xml-binary(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
defetch:xml-binary(convert:string-to-base64(
  "<xml version='1.0' encoding='CP1252'?><xml>touché</xml>",
  "CP1252"
))
```
|                     | • Encodes a string as UTF16 and parses it as XML. The document will be correctly decoded, as the first bytes of the data indicate that the input must be UTF16:
**fetch:content-type**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>fetch:content-type($uri as xs:string) as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the content-type (also called mime-type) of the resource specified by $uri:</td>
</tr>
<tr>
<td></td>
<td>• If a remote resource is addressed, the request header will be evaluated.</td>
</tr>
<tr>
<td></td>
<td>• If the addressed resource is locally stored, the content-type will be guessed based on the file extension.</td>
</tr>
<tr>
<td>Errors</td>
<td>open: the URI could not be resolved, or the resource could not be retrieved.</td>
</tr>
<tr>
<td>Examples</td>
<td>• fetch:content-type(&quot;<a href="http://docs.basex.org/skins/vector/images/wiki.png">http://docs.basex.org/skins/vector/images/wiki.png</a>&quot;) returns image/png.</td>
</tr>
</tbody>
</table>

**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 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.

### 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**

```xml
file:list($dir as xs:string) as xs:string*
file:list($dir as xs:string, $recursive as xs:boolean) as xs:string*
file:list($dir as xs:string, $recursive as xs:boolean, $pattern as xs:string) as xs:string*
```

**Summary**

Lists all files and directories found in the specified `$dir`. The returned paths are relative to the provided path. The optional parameter `$recursive` specifies whether sub-directories will be traversed, too. The optional parameter `$pattern` defines a file name pattern in the Glob Syntax. If present, only those files and directories are returned that correspond to the pattern. Several patterns can be separated with a comma (,).

**Errors**

- `not-found`: the specified file does not exist.
- `no-dir`: the specified path does not point to a directory.
- `io-error`: the operation fails for some other reason.

**file:children**

**Signatures**

```xml
file:children($dir as xs:string) as xs:string*
```

**Summary**

Returns the full paths to all files and directories found in the specified `$dir`. The inverse function is `file:parent`. The related function `file:list` returns relative file paths.

**Errors**

- `not-found`: the specified file does not exist.
- `no-dir`: the specified path does not point to a directory.
- `io-error`: the operation fails for some other reason.

**file:read-binary**

**Signatures**

```xml
file:read-binary($path as xs:string) as xs:base64Binary
file:read-binary($path as xs:string, $offset as xs:integer) as xs:base64Binary
file:read-binary($path as xs:string, $offset as xs:integer, $length as xs:integer) as xs:base64Binary
```
### File Module

**Summary**

Reads the binary content of the file specified by $path and returns it as lazy xs:base64Binary item. The optional parameters $offset and $length can be used to read chunks of a file.

**Errors**

- `not-found`: the specified file does not exist.
- `is-dir`: the specified path is a directory.
- `out-of-range`: the offset or length is negative, or the chosen values would exceed the file bounds.
- `io-error`: the operation fails for some other reason.

**Examples**

- `lazy:cache(file:read-binary("config.data"))` enforces the file access (otherwise, it will be delayed until requested first).

#### file:read-text

**Signatures**

```
file:read-text($path as xs:string) as xs:string
file:read-text($path as xs:string, $encoding as xs:string) as xs:string
file:read-text($path as xs:string, $encoding as xs:string, $fallback as xs:boolean) as xs:string
```

**Summary**

Reads the textual contents of the file specified by $path 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 U+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

**Signatures**

```
file:read-text-lines($path as xs:string) as xs:string*
file:read-text-lines($path as xs:string, $encoding as xs:string) as xs:string*
file:read-text-lines($path as xs:string, $encoding as xs:string, $fallback as xs:boolean) as xs:string*
file:read-text-lines($path as xs:string, $encoding as xs:string, $fallback as xs:boolean, $offset as xs:integer) as xs:string*
file:read-text-lines($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 U+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()
```
File Module

<table>
<thead>
<tr>
<th>Summary</th>
<th>Creates the directory specified by $dir, including all non-existing parent directories.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Errors</td>
<td>exists: a file with the same path already exists.io-error: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**file:create-temp-dir**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>no-dir: the specified directory points to a file.io-error: the directory could not be created.</td>
</tr>
</tbody>
</table>

**file:create-temp-file**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>no-dir: the specified directory points to a file.io-error: the directory could not be created.</td>
</tr>
</tbody>
</table>

**file:delete**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:delete($path as xs:string) as empty-sequence() file:delete($path as xs:string, $recursive as xs:boolean) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Recursively deletes a file or directory specified by $path. The optional parameter $recursive specifies whether sub-directories will be deleted, too.</td>
</tr>
<tr>
<td>Errors</td>
<td>not-found: the specified path does not exist.io-error: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**file:write**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>file:write($path as xs:string, $items as item()<em>) as empty-sequence() file:write($path as xs:string, $items as item()</em>, $params as item()) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Writes a serialized sequence of items to the specified file. If the file already exists, it will be overwritten. The $params argument contains serialization parameters (see Serialization for more details), which can either be specified</td>
</tr>
</tbody>
</table>

- as children of an `<output:serialization-parameters/>` element, as defined for the `fn:serialize()` function; e.g.:

```xml
<output:serialization-parameters>
  <output:method value='xml'/>
  <output:cdata-section-elements value="div"/>
  ...
</output:serialization-parameters>
```
File Module

```xml
<serialization-parameters>
  
  * as map, which contains all key/value pairs:

  map { "method": "xml", "cdata-section-elements": "div", ... }

Errors

- no-dir: the parent of specified path is no directory.
- is-dir: the specified path is a directory.
- io-error: the operation fails for some other reason.

**file:write-binary**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

**file:write-text**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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).</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

**file:write-text-lines**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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).</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>

**file:append**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Appends a serialized sequence of items to the specified file. If the file does not exists, a new file is created.</td>
</tr>
<tr>
<td>Errors</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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:copy

**Signatures**
```
file:copy($source as xs:string, $target as xs:string) as empty-sequence()
```

**Summary**
Copies a file or directory specified by $source to the file or directory specified by $target. If the target file already exists, it will be overwritten. No operation will be performed if the source and target path are equal.

**Errors**
- **not-found**: the specified source does not exist.
- **exists**: the specified source is a directory and the target is a file.
- **no-dir**: the parent of the specified target is no directory.
- **io-error**: the operation fails for some other reason.

### file:move

**Signatures**
```
file:move($source as xs:string, $target as xs:string) as empty-sequence()
```

**Summary**
Moves or renames the file or directory specified by $source to the path specified by $target. If the target file already exists, it will be overwritten. No operation will be performed if the source and target path are equal.

**Errors**
- **not-found**: the specified source does not exist.
- **exists**: the specified source is a directory and the target is a file.
- **no-dir**: the parent of the specified target is no directory.
- **io-error**: the operation fails for some other reason.
File Properties

file:exists

Signatures
file:exists($path as xs:string) as xs:boolean

Summary
Returns an xs:boolean indicating whether a file or directory specified by $path exists in the file system.

file:is-dir

Signatures
file:is-dir($path as xs:string) as xs:boolean

Summary
Returns an xs:boolean indicating whether the argument $path points to an existing directory.

file:is-absolute

Signatures
file:is-absolute($path as xs:string) as xs:boolean

Summary
Returns an xs:boolean indicating whether the argument $path is absolute. The behavior of this function depends on the operating system: On Windows, an absolute path starts with the drive letter and a colon, whereas on Linux it starts with a slash.

file:is-file

Signatures
file:is-file($path as xs:string) as xs:boolean

Summary
Returns an xs:boolean indicating whether the argument $path points to an existing file.

file:last-modified

Signatures
file:last-modified($path as xs:string) as xs:dateTime

Summary
Retrieves the timestamp of the last modification of the file or directory specified by $path.

Errors
not-found: the specified path does not exist.

file:size

Signatures
file:size($file as xs:string) as xs:integer

Summary
Returns the size, in bytes, of the file specified by $path, or 0 for directories.

Errors
not-found: the specified file does not exist.

Path Functions

file:name

Signatures
file:name($path as xs:string) as xs:string

Summary
Returns the name of a file or directory specified by $path. An empty string is returned if the path points to the root directory.

file:parent

Signatures
file:parent($path as xs:string) as xs:string?

Summary
Returns the absolute path to the parent directory of a file or directory specified by $path. An empty sequence is returned if the path points to a root directory. The inverse function is file:children.
Examples

• file:parent(static-base-uri()) returns the directory of the current XQuery module.

file:path-to-native

Signatures

file:path-to-native($path as xs:string) as xs:string

Summary

Transforms the $path argument to its native representation on the operating system.

Errors

not-found: the specified file does not exist.
io-error: the specified path cannot be transformed to its native representation.

file:resolve-path

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 \n, #10, #9, #13, or #10;

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 9.0

- 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


Version 7.8

- Added: `file:parent`, `file:name`
- Updated: error codes; `file:read-binary`, `file:write-binary`: $offset and $length arguments added.
File Module

• Deleted: file:base-name, file:dir-name

Version 7.7

• Added: file:create-temp-dir, file:create-temp-file, file:temp-dir

• Updated: all returned strings that refer to existing directories will be suffixed with a directory separator.

Version 7.3

• Added: file:append-text, file:write-text, file:append-text-lines, file:write-text-lines, file:line-separator

• Aligned with latest specification: $file:directory-separator \rightarrow file:dir-separator, $file:path-separator \rightarrow file:path-separator, file:is-directory \rightarrow file:is-dir, file:create-directory \rightarrow file:create-dir

• Updated: file:write-binary, file:append-binary: output limited to a single value

Version 7.2.1

• Updated: file:delete: $recursive parameter added to prevent sub-directories from being accidentally deleted.

• Fixed: file:list now returns relative instead of absolute paths.
Chapter 45. Full-Text Module

Read this entry online in the BaseX Wiki.

This XQuery Module extends the W3C Full Text Recommendation with some useful functions: The index can be directly accessed, fulltext 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.

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**: determines the mode how tokens are searched. Allowed values are any, any word, all, all words, and phrase. any is the default search mode.
- **fuzzy**: turns fuzzy querying on or off. Allowed values are true and false. By default, fuzzy querying is turned off.
- **wildcards**: turns wildcard querying on or off. Allowed values are true and false. By default, wildcard querying is turned off.
- **ordered**: requires that all tokens occur in the order in which they are specified. Allowed values are true and false. The default is false.
- **content**: specifies that the matched tokens need to occur at the beginning or end of a searched string, or need to cover the entire string. Allowed values are start, end, and entire. By default, the option is turned off.
- **scope**: defines the scope in which tokens must be located. The option has following sub options:
  - **same**: can be set to true or false. It specifies if tokens need to occur in the same or different units.
  - **unit**: can be sentence or paragraph. It specifies the unit for finding tokens.
  - **window**: sets up a window in which all tokens must be located. By default, the option is turned off. It has following sub options:
    - **size**: specifies the size of the window in terms of units.
    - **unit**: can be sentences, sentences or paragraphs. The default is words.
• **distance**: specifies the distance in which tokens must occur. By default, the option is turned off. It has following sub options:
  
  • **min**: specifies the minimum distance in terms of units. The default is 0.
  
  • **max**: specifies the maximum distance in terms of units. The default is #.
  
  • **unit**: can be words, sentences or paragraphs. The default is words.

**Errors**

- **db:open**: 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: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 in addition:

- **case**: determines how character case is processed. Allowed values are insensitive, sensitive, upper and lower. By default, search is case insensitive.
- **diacritics**: determines how diacritical characters are processed. Allowed values are insensitive and sensitive. By default, search is diacritical insensitive.
- **stemming**: determines is tokens are stemmed. Allowed values are true and false. By default, stemming is turned off.
- **language**: determines the language. This option is relevant for stemming tokens. All language codes are supported. The default language is en.

**Errors**

- options: specified options are conflicting.

**Examples**

- Checks if jack or john occurs in the input string John Doe:
ft:contains("John Doe", ("jack", "john"), map { "mode": "any" })

- Calls the function with stemming turned on and off:

(true(), false()) ! ft:contains("Häuser", "Haus", map { 'stemming': ., 'language':'de' })

### ft:mark

**Signatures**

| ft:mark($nodes as node()) as node()* | ft:mark($nodes as node()*, $name as xs:string) as node()* |

**Summary**

Puts a marker element around the resulting $nodes of a full-text index request. The default name of the marker element is mark. An alternative name can be chosen via the optional $name argument. Please note that:

- the full-text expression that computes the token positions must be specified as argument of the ft:mark() function, as all position information is lost in subsequent processing steps. You may need to specify more than one full-text expression if you want to use the function in a FLWOR expression, as shown in Example 2.

- the XML node to be transformed must be an internal "database" node. The transform expression can be used to apply the method to a main-memory fragment, as shown in Example 3.

**Examples**

**Example 1**: The following query returns <XML><mark>hello</mark> world</XML>, if one text node of the database DB has the value "hello world":

```xml
ft:mark(db:open('DB')//*[text() contains text 'hello'])
```

**Example 2**: The following expression loops through the first ten full-text results and marks the results in a second expression:

```xml
let $start := 1
let $end   := 10
let $term  := 'welcome'
for $ft in (db:open('DB')//*[text() contains text { $term }])[position() = $start to $end]
return element hit {
  ft:mark($ft[text() contains text { $term }])
}
```

**Example 3**: The following expression returns <p><b>word</b></p>:  

```xml
copy $p := <p>word</p>
modify ()
return ft:mark($p[text() contains text 'word'], 'b')
```

### ft:extract

**Signatures**

| ft:extract($nodes as node()) as node()* | ft:extract($nodes as node()*, $name as xs:string) as node()* | ft:extract($nodes as node()*, $name as xs:string, $length as xs:integer) as node()* |

**Summary**

Extracts and returns relevant parts of full-text results. It puts a marker element around the resulting $nodes of a full-text index request and chops irrelevant sections of the result. The default 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:open('DB')//*[text() contains text 'hello'], 'b', 1)
  ```

### `ft:count`

**Signatures**

```xml
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**

```xml
ft:score($item as item()) as xs:double*
```

**Summary**

Returns the score values (0.0 - 1.0) that have been attached to the specified items. 0 is returned a value if no score was attached.

**Examples**
- `ft:score('a' contains text 'a')` returns the `xs:double` value 1.

### `ft:tokens`

**Signatures**

```xml
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:open`: 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:

  ```xml
  let $term := ft:tokenize($term)
  return number(ft:tokens('db', $term)[. = $term]/@count)
  ```

### `ft:tokenize`

Updated with Version 9.1: support for empty sequence.

**Signatures**

```xml
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

Updated with Version 9.1: support for empty sequence.

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 expects the same arguments as ft:tokenize.

Examples
• ft:tokenize("Häuser am Meer", map { 'case': 'sensitive' }) returns the string Hauser am Meer.

Errors

Code Description
options Both wildcards and fuzzy search have been specified as search options.

Changelog

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)

238
Version 7.1

- Added: ft:tokens, ft:tokenize
Chapter 46. Geo Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions that may be applied to geometry data conforming to the Open Geospatial Consortium (OGC) Simple Feature (SF) data model. It is based on the EXPath Geo Module and uses the JTS library.

Geometries included in GML 2 are: Point, LineString, LinearRing, Polygon, MultiPoint, MultiLineString, MultiPolygon, and MultiGeometry. All nodes queried by BaseX should be a valid geometry. The only geometry type which is not supported by BaseX right now is MultiGeometry. Moreover, the module provides no support for GML 3.

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://expath.org/ns/geo namespace. The module must be imported in the query prolog:

```
import module namespace geo = "http://expath.org/ns/geo";
...
```
- In this document, the namespace is bound to the geo prefix.
- All errors are assigned to the http://expath.org/ns/error namespace, which is statically bound to the experr prefix.

General Functions

**geo:dimension**

**Signatures**

```
geo:dimension($geometry as element(*)) as xs:integer
```

**Summary**

Returns the dimension of the given geometry $geometry.

**Errors**

GEO0001: the given element is not recognized as a valid geometry.GEO0002: the given element cannot be read by reader for some reason.

**Example**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:dimension(<gml:Point><gml:coordinates>1,2</gml:coordinates></gml:Point>)
```

**geo:geometry-type**

**Signatures**

```
geo:geometry-type($geometry as element(*)) as xs:QName
```

**Summary**

Returns the name of the geometry type of given geometry $geometry, if the geometry is not recognized with an error massage.

**Errors**

GEO0001: the given element is not recognized as a valid geometry.GEO0002: the given element cannot be read by reader for some reason.

**Example**

```
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:geometry-type(<gml:Point><gml:coordinates>1,2</gml:coordinates></gml:Point>)
```
Geo Module

**geo:srid**

**Signatures**

```
geo:srid($geometry as element(*)) as xs:integer
```

**Summary**

Returns the ID of the Spatial Reference System used by the given geometry $geometry. Spatial Reference System information is supported in the simple way defined in the SFS. A Spatial Reference System ID (SRID) is present in each Geometry object. Geometry provides basic accessor operations for this field, but no others. The SRID is represented as an integer (based on the OpenGIS Simple Features Specifications For SQL). Here is a difference between the EXPath Geo Module and the implementation in BaseX, since the specification return the URI.

**Errors**

GEO0001: the given element is not recognized as a valid geometry. GEO0002: the given element cannot be read by reader for some reason.

**Examples**

```xml
import module namespace geo='http://expath.org/ns/geo';
declare namespace gml='http://www.opengis.net/gml';
geo:srid(
  <gml:Polygon>
    <outerboundaryIs>
      <gml:LinearRing>
        <coordinates>-150,50 -150,60 -125,60 -125,50 -150,50</coordinates>
      </gml:LinearRing>
    </outerboundaryIs>
  </gml:Polygon>
)
```

**geo:envelope**

**Signatures**

```
geo:envelope($geometry as element(*)) as element(*)
```

**Summary**

Returns the gml:Envelope of the given geometry $geometry. The envelope is the minimum bounding box of this geometry. If this Geometry is the empty geometry, returns an empty Point. If the Geometry is a point, returns a non-empty Point. Otherwise, returns a Polygon whose points are (minx, miny), (maxx, miny), (minx, maxy), (maxx, maxy), (minx, miny).

**Errors**

GEO0001: the given element is not recognized as a valid geometry. GEO0002: the given element cannot be read by reader for some reason. GEO0005: the output object cannot be written as an element by writer for some reason.

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:envelope(
  <gml:LinearRing>
    <gml:coordinates>1,1 20,1 20,20 1,20 1,1</gml:coordinates>
  </gml:LinearRing>
)
```

**geo:as-text**

**Signatures**

```
geo:as-text($geometry as element(*)) as xs:string
```

**Summary**

Returns the WKT (Well-known Text) representation of the given geometry $geometry. The envelope is the minimum bounding box of this geometry.

**Errors**

GEO0001: the given element is not recognized as a valid geometry. GEO0002: the given element cannot be read by reader for some reason.

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:as-text(<gml:Point><gml:coordinates>1,2</gml:coordinates></gml:Point>)
```
### geo:as-binary

**Signatures**
geo:as-binary($geometry as element(*)) as xs:base64Binary

**Summary**
Returns the WKB (Well-known Binary) representation of the given geometry $geometry.

**Errors**
GEO0001: the given element is not recognized as a valid geometry.
GEO0002: the given element cannot be read by reader for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:as-binary(<gml:Point><gml:coordinates>1,2</gml:coordinates></gml:Point>)
```

### geo:is-simple

**Signatures**
geo:is-simple($geometry as element(*)) as xs:boolean

**Summary**
Returns whether the given geometry is simple $geometry and does not have has no anomalous
geometric points (ie. the geometry does not self-intersect and does not pass through the same point
more than once (may be a ring)).

**Errors**
GEO0001: the given element is not recognized as a valid geometry.
GEO0002: the given element cannot be read by reader for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
```

### geo:boundary

**Signatures**
geo:boundary($geometry as element(*)) as element(*)?

**Summary**
Returns the boundary of the given geometry $geometry, in GML 2. The return value is a sequence
of either gml:Point or gml:LinearRing elements as a GeometryCollection object. For a Point or
MultiPoint, the boundary is the empty geometry, nothing is returned.

**Errors**
GEO0001: the given element is not recognized as a valid geometry.
GEO0002: the given element cannot be read by reader for some reason.
GEO0005: the output object cannot be written as an
element by writer for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:boundary(<gml:LineString><gml:coordinates>1,1 0,0 2,1</gml:coordinates></gml:LineString>)
```

### geo:num-geometries

**Signatures**
geo:num-geometries($geometry as element(*)) as xs:integer

**Summary**
Returns the number of geometry in a geometry-collection $geometry, in GML. For the
geometries which are not a collection, it returns the instant value 1. This function is implemented
Geo Module

wider than the specification and accepts all types of geometries, while the specification limits it to the collection types (MultiPoint, MultiPolygon, ...).

<table>
<thead>
<tr>
<th>Errors</th>
<th>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.</th>
</tr>
</thead>
</table>
| Example         | ```import module namespace geo = "http://expat.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:num-geometries(
    <gml:MultiLineString>
        <gml:LineString><gml:coordinates>1,1 0,0 2,1</gml:coordinates></gml:LineString>
        <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
    </gml:MultiLineString>
)
``` |

geo:geometry-n

<table>
<thead>
<tr>
<th>Signatures</th>
<th>geo:geometry-n($geometry as element(<em>), $geoNumber as xs:integer) as element(</em>)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the Nth geometry in geometry-collection $geometry, in GML. For the geometries which are not a collection, it returns the geometry if geoNumber $geoNumber is 1. This function is implemented wider than the specification and accepts all types of geometries, while the specification limits it to the collection types (MultiPoint, MultiPolygon, ...).</td>
</tr>
<tr>
<td>Errors</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0004: the input index of geometry is out of range. GEO0005: the output object cannot be written as an element by writer for some reason.</td>
</tr>
</tbody>
</table>
| Example         | ```import module namespace geo = "http://expat.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
    <gml:MultiLineString>
        <gml:LineString><gml:coordinates>1,1 0,0 2,1</gml:coordinates></gml:LineString>
        <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
    </gml:MultiLineString>
return geo:geometry-n($line, 1)``` |

geo:length

<table>
<thead>
<tr>
<th>Signatures</th>
<th>geo:length($geometry as element(*)) as xs:double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the length of the geometry $geometry. If the geometry is a point, zero value will be returned.</td>
</tr>
<tr>
<td>Errors</td>
<td>GEO00001: the given element(s) is not recognized as a valid geometry (QName). GEO00002: the given element cannot be read by reader for some reason.</td>
</tr>
</tbody>
</table>
| Example         | ```import module namespace geo = "http://expat.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon :=
    <gml:Polygon>
        <gml:outerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
            </gml:LinearRing>
        </gml:outerBoundaryIs>
    </gml:Polygon>
return geo:length($polygon)``` |
Geo Module

geo:num-points

Signatures
geo:num-points($geometry as element(*)) as xs:integer

Summary
Returns integer value of number of the points in the given geometry $geometry. It can be used not only for Lines, also any other geometry types, like MultiPolygon. For Point geometry it will return 1. This is an implementation different from the XPath geo specification, as it limits the input geometry type only to lines.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:num-points(
  <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
)

geo:area

Signatures
geo:area($geometry as element(*)) as xs:double

Summary
Returns the area of the given geometry $geometry. For points and line the return value will be zero.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
return geo:area($polygon)

geo:centroid

Signatures
geo:centroid($geometry as element(*)) as element(*)

Summary
Returns the mathematical centroid of the given geometry $geometry, as a gml:Point. Based on the definition, this point is not always on the surface of the geometry $geometry.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0005: the output object cannot be written as an element by writer for some reason.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $point :=
  <gml:MultiPoint>
    <gml:Point><gml:coordinates>1,1</gml:coordinates></gml:Point>
    <gml:Point><gml:coordinates>10,10</gml:coordinates></gml:Point>
    <gml:Point><gml:coordinates>2,2</gml:coordinates></gml:Point>
  </gml:MultiPoint>
Geo Module

geo:point-on-surface

**Signatures**
geo:point-on-surface($geometry as element(*)) as element(*)

**Summary**
Returns an interior point on the given geometry $geometry, as a gml:Point. It is guaranteed to be on surface. Otherwise, the point may lie on the boundary of the geometry.

**Errors**
GEO0001: the given element(s) is not recognized as a valid geometry (QName).
GEO0002: the given element cannot be read by reader for some reason.
GEO0005: the output object cannot be written as an element by writer for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:point-on-surface(
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>1,1 2,1 2,2 1,2 1,1</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
)
```

Spatial Predicate Functions

geo:equals

**Signatures**
geo:equals($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean

**Summary**
Returns whether geometry1 $geometry1 is spatially equal to $geometry2 $geometry2.

**Errors**
GEO0001: the given element(s) is not recognized as a valid geometry (QName).
GEO0002: the given element cannot be read by reader for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:equals(
  <gml:LineString><gml:coordinates>1,1 55,99 2,1</gml:coordinates></gml:LineString>,
  <gml:LineString><gml:coordinates>1,1 1,1 55,99 2,1</gml:coordinates></gml:LineString>
)
```

geo:disjoint

**Signatures**
geo:disjoint($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean

**Summary**
Returns whether geometry1 $geometry1 is spatially disjoint from $geometry2 $geometry2 (they have no point in common, they do not intersect each other, and the DE-9IM Intersection Matrix for the two geometries is FF*FF****).

**Errors**
GEO0001: the given element(s) is not recognized as a valid geometry (QName).
GEO0002: the given element cannot be read by reader for some reason.

**Example**
```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $lines :=
  <gml:MultiLineString>
```
geo:intersect

Signatures
geo:intersects($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean

Summary
Returns whether $geometry1 is spatially intersects $geometry2. This is true if disjoint function of the two geometries returns false.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $lines :=
  <gml:MultiLineString>
    <gml:LineString><gml:coordinates>1,1 0,0 2,1</gml:coordinates></gml:LineString>
    <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
  </gml:MultiLineString>
let $line :=
  <gml:LineString><gml:coordinates>0,0 2,1 3,3</gml:coordinates></gml:LineString>
return geo:intersects($lines, $line)

geo:touches

Signatures
geo:touches($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean

Summary
Returns whether $geometry1 is spatially touches $geometry2.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
  <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
return geo:touches($line, $polygon)

geo:crosses

Signatures
geo:crosses($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean
<table>
<thead>
<tr>
<th>Geo Module</th>
</tr>
</thead>
</table>

### Summary
| | Returns whether geometry1 $geometry1 is spatially crosses $geometry2 $geometry2. It means, if the geometries have some but not all interior points in common. Returns true if the DE-9IM intersection matrix for the two geometries is: 
| | T*T***** (for P/L, P/A, and L/A situations)
| | T*****T** (for L/P, A/P, and A/L situations)
| | 0******** (for L/L situations). |

### Errors
| | GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the given element cannot be read by reader for some reason. |

#### Example
| | import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
        <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
let $polygon :=
        <gml:Polygon>
            <gml:outerBoundaryIs>
                <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
            </gml:outerBoundaryIs>
        </gml:Polygon>
return geo:crosses($line, $polygon) |

#### geo:within
| Signatures | geo:within($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean |
| Summary | Returns whether geometry1 $geometry1 is spatially within $geometry2 $geometry2. |
| Errors | GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the given element cannot be read by reader for some reason. |

#### Example
| | import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
        <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
let $polygon :=
        <gml:Polygon>
            <gml:outerBoundaryIs>
                <gml:LinearRing><gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates></gml:LinearRing>
            </gml:outerBoundaryIs>
        </gml:Polygon>
return geo:within($line, $polygon) |

#### geo:contains
| Signatures | geo:contains($geometry1 as element(*), $geometry2 as element(*)) as xs:boolean |
| Summary | Returns whether geometry1 $geometry1 spatially contains $geometry2 $geometry2. Returns true if within function of these two geometries also returns true. |
| Errors | GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the given element cannot be read by reader for some reason. |

#### Example
| | import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $point :=
        <gml:Point><gml:coordinates>1,1</gml:coordinates></gml:Point>
let $polygon :=
        <gml:Polygon>
Geo Module

geo:contains

Signatures geo:contains($polygon, $point)
as xs:boolean

Summary Returns whether the point $point
is contained in the polygon $polygon.

Errors GEO0001: the given element(s) is not recognized as a valid geometry (QName).

Example

import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon1 :=
    <gml:Polygon>
        <gml:outerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>1,1 20,1 20,20 30,20 30,30 1,30 1,1</gml:coordinates>
            </gml:LinearRing>
        </gml:outerBoundaryIs>
        <gml:innerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>2,2 3,2 3,3 2,3 2,2</gml:coordinates>
            </gml:LinearRing>
        </gml:innerBoundaryIs>
        <gml:innerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>10,10 19,10 19,19 10,19 10,10</gml:coordinates>
            </gml:LinearRing>
        </gml:innerBoundaryIs>
    </gml:Polygon>
let $polygon2 :=
    <gml:Polygon>
        <gml:outerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates>
            </gml:LinearRing>
        </gml:outerBoundaryIs>
    </gml:Polygon>
return geo:contains($polygon1, $point)

geo:overlaps

Signatures geo:overlaps($geometry1 as element(*), $geometry2 as element(*))
as xs:boolean

Summary Returns whether geometry1 $geometry1
is spatially overlaps $geometry2 $geometry2.

Errors GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the
given element cannot be read by reader for some reason.

Example

import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon1 :=
    <gml:Polygon>
        <gml:outerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates>
            </gml:LinearRing>
        </gml:outerBoundaryIs>
    </gml:Polygon>
let $polygon2 :=
    <gml:Polygon>
        <gml:outerBoundaryIs>
            <gml:LinearRing>
                <gml:coordinates>1,1 2,1 5,3 1,1</gml:coordinates>
            </gml:LinearRing>
        </gml:outerBoundaryIs>
    </gml:Polygon>
return geo:overlaps($polygon1, $polygon2)

geo:relate

Signatures geo:relate($geometry1 as element(*), $geometry2 as element(*),
$intersectionMatrix as xs:string) as xs:boolean

Summary Returns whether relationships between the boundaries, interiors and exteriors of geometry1
$geometry1 and geometry2 $geometry2 match the pattern specified in intersectionMatrix
$intersectionMatrix, which should have the length of 9 charachters.The values in the DE-9IM can be
T, F, *, 0, 1, 2 . - T means the intersection gives a non-empty result. - F means the intersection
gives an empty result. - * means any result. - 0, 1, 2 gives the expected dimension of the result
(point, curve, surface)

Errors GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the
given element cannot be read by reader for some reason.GEO0006: the given matrix is invalid.

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $point :=
  <gml:Point><gml:coordinates>18,11</gml:coordinates></gml:Point>

let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>10,10 20,10 30,40 20,40 10,10</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>

return geo:relate($point, $polygon)

Analysis Functions

geo:distance

Signatures
geo:distance($geometry1 as element(*), $geometry2 as element(*)) as xs:double

Summary
Returns the shortest distance, in the units of the spatial reference system of geometry1 $geometry1, between the geometries, where that distance is the distance between a point on each of the geometries.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName).

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';

let $line :=
  <gml:LinearRing>
    <gml:coordinates>10,400 20,200 30,100 20,100 10,400</gml:coordinates>
  </gml:LinearRing>

let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>10,10 20,10 30,40 20,40 10,10</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>

return geo:distance($line, $polygon)

geo:buffer

Signatures
geo:buffer($geometry as element(*), $distance as xs:double) as element(*)

Summary
Returns polygonal geometry representing the buffer by distance $distance of geometry $geometry, a buffer area around this geometry having the given width, in the spatial reference system of geometry. The buffer of a Geometry is the Minkowski sum or difference of the geometry with a disc of radius abs(distance). The buffer is constructed using 8 segments per quadrant to represent curves.

Errors
GEO0001: the given element(s) is not recognized as a valid geometry (QName).

Example
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon :=
<gml:Polygon>
  <gml:outerBoundaryIs>
    <gml:LinearRing>
      <gml:coordinates>10,10 20,10 30,40 20,40 10,10</gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
</gml:Polygon>
return geo:buffer($polygon, 3)

geo:convex-hull

Signatures  geo:convex-hull($geometry as element(*)) as element(*)

Summary  Returns the convex hull geometry of the given geometry $geometry in GML, or the empty sequence. Actually returns the object of smallest dimension possible.

Errors  GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0005: the output object cannot be written as an element by writer for some reason.

Example  import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:convex-hull(
  <gml:LinearRing>
    <gml:coordinates>10,400 20,200 30,100 20,100 10,400</gml:coordinates>
  </gml:LinearRing>
)

geo:intersection

Signatures  geo:intersection($geometry1 as element(*), $geometry2 as element(*)) as element(*)?

Summary  Returns the intersection geometry of geometry1 $geometry1 with geometry2 $geometry2, in GML or empty sequence if there is no intersection of these geometries.

Errors  GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0005: the output object cannot be written as an element by writer for some reason.

Example  import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
  <gml:LinearRing>
    <gml:coordinates>10,400 20,200 30,100 20,100 10,400</gml:coordinates>
  </gml:LinearRing>
let $point :=
  <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></gml:Point>
return geo:intersection($line, $point)

geo:union

Signatures  geo:union($geometry1 as element(*), $geometry2 as element(*)) as element(*)

Summary  Returns the union geometry of geometry1 $geometry1 with geometry2 $geometry2, in GML.

Errors  GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0005: the output object cannot be written as an element by writer for some reason.
**Geo Module**

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $line :=
  <gml:LinearRing>
    <gml:coordinates>10,400 20,200 30,100 20,100 10,400</gml:coordinates>
  </gml:LinearRing>
let $point :=
  <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></gml:Point>
return geo:union($line, $point)
```

**geo:difference**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>geo:difference($geometry1 as element(<em>), $geometry2 as element(</em>)) as element(*)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the difference geometry of geometry1 $geometry1 with geometry2 $geometry2, in GML, or empty sequence if the difference is empty, as a set of point in geometry1 $geometry1 and not included in geometry2 $geometry2.</td>
</tr>
<tr>
<td>Errors</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the given element cannot be read by reader for some reason.GEO0005: the output object cannot be written as an element by writer for some reason.</td>
</tr>
</tbody>
</table>

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $point :=
  <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></gml:Point>
let $line :=
  <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
return geo:difference($point, $line)
```

**geo:sym-difference**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>geo:sym-difference($geometry1 as element(<em>), $geometry2 as element(</em>)) as element(*)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the symmetric difference geometry of geometry1 $geometry1 with geometry2 $geometry2, in GML, or empty sequence if the difference is empty, as a set of point in one of the geometries and not included in the other.</td>
</tr>
<tr>
<td>Errors</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName).GEO0002: the given element cannot be read by reader for some reason.GEO0005: the output object cannot be written as an element by writer for some reason.</td>
</tr>
</tbody>
</table>

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $point :=
  <gml:Point><gml:coordinates>1.00,1.00</gml:coordinates></gml:Point>
let $line :=
  <gml:LineString><gml:coordinates>2,1 3,3 4,4</gml:coordinates></gml:LineString>
return geo:sym-difference($point, $line)
```

**Functions Specific to Geometry Type**

**geo:x**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>geo:x($point as element(*)) as xs:double</th>
</tr>
</thead>
</table>

---

251
<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
<th>Summary</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>geo:x()</code></td>
<td><code>geo:x($point as element(*)) as xs:double?</code></td>
<td>Returns the x coordinate of point $point. A point has to have an x coordinate.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.</td>
</tr>
<tr>
<td><code>geo:y()</code></td>
<td><code>geo:y($point as element(*)) as xs:double?</code></td>
<td>Returns the y coordinate of point $point. If the point does not have the y coordinate, 0 will be returned.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.</td>
</tr>
<tr>
<td><code>geo:z()</code></td>
<td><code>geo:z($point as element(*)) as xs:double?</code></td>
<td>Returns the z coordinate of point $point. If the point does not have the z coordinate, 0 will be returned.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason.</td>
</tr>
<tr>
<td><code>geo:start-point()</code></td>
<td><code>geo:start-point($line as element(*)) as element(*)</code></td>
<td>Returns the starting point of the given line $line. $line has to be a single line, LineString or LinearRing.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0003: the given element has to be a line. Other geometries are not accepted.</td>
</tr>
<tr>
<td><code>geo:end-point()</code></td>
<td><code>geo:end-point($line as element(*)) as element(*)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Geo Module

<table>
<thead>
<tr>
<th>Function</th>
<th>Signatures</th>
<th>Summary</th>
<th>Errors</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo:end-point</td>
<td>geo:end-point($line as element(*)) as xs:string</td>
<td>Returns the ending point of the given line $line. $line has to be a single line, LineString or LinearRing.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0003: the given element has to be a line. Other geometries are not accepted.</td>
<td>import module namespace geo = &quot;<a href="http://expath.org/ns/geo">http://expath.org/ns/geo</a>&quot;; declare namespace gml='<a href="http://www.opengis.net/gml">http://www.opengis.net/gml</a>'; geo:end-point( <a href="">gml:LineString</a><a href="">gml:coordinates</a>2,1 3,3 4,4&lt;/gml:coordinates&gt;&lt;/gml:LineString&gt; )</td>
</tr>
<tr>
<td>geo:is-closed</td>
<td>geo:is-closed($line as element(*)) as xs:boolean</td>
<td>Returns a boolean value that shows the line $line is a closed loop (start point and end point are the same) or not. $line has to be a line, as a geometry, LineString or LinearRing.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0003: the given element has to be a line. Other geometries are not accepted.</td>
<td>import module namespace geo = &quot;<a href="http://expath.org/ns/geo">http://expath.org/ns/geo</a>&quot;; declare namespace gml='<a href="http://www.opengis.net/gml">http://www.opengis.net/gml</a>'; geo:is-closed( <a href="">gml:LineString</a> <a href="">gml:coordinates</a>2,1 3,3 4,4&lt;/gml:coordinates&gt; &lt;/gml:LineString&gt; )</td>
</tr>
<tr>
<td>geo:is-ring</td>
<td>geo:is-ring($line as element(*)) as xs:boolean</td>
<td>Returns a boolean value that shows the line $line is a ring (closed loop and single) or not. $line has to be a single line, as a geometry, LineString or LinearRing.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0003: the given element has to be a line. Other geometries are not accepted.</td>
<td>import module namespace geo = &quot;<a href="http://expath.org/ns/geo">http://expath.org/ns/geo</a>&quot;; declare namespace gml='<a href="http://www.opengis.net/gml">http://www.opengis.net/gml</a>'; geo:is-ring( <a href="">gml:LineString</a> <a href="">gml:coordinates</a>2,1 3,3 4,4&lt;/gml:coordinates&gt; &lt;/gml:LineString&gt; )</td>
</tr>
<tr>
<td>geo:point-n</td>
<td>geo:point-n($line as element(<em>)) as element(</em>)</td>
<td>Returns the Nth point in the given line $geometry. $line has to be a single line, as a geometry, LineString or LinearRing.</td>
<td>GEO0001: the given element(s) is not recognized as a valid geometry (QName). GEO0002: the given element cannot be read by reader for some reason. GEO0003: the given element has to be a line. Other geometries are not accepted. GEO0004: the the input index of geometry is out of range. GEO0005: the output object cannot be written as an element by writer for some reason.</td>
<td></td>
</tr>
</tbody>
</table>
### geo:exterior-ring

**Signatures**

```
geo:exterior-ring($polygon as element( *)) as element ( *)
```

**Summary**

Returns the outer ring of the given polygon $geometry, as a gml:LineString.

**Errors**

- GEO0001: the given element(s) is not recognized as a valid geometry (QName).
- GEO0002: the given element cannot be read by reader for some reason.
- GEO0003: the given element has to be a polygon. Other geometries are not accepted.
- GEO0005: the output object cannot be written as an element by writer for some reason.

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>10,10 20,10 30,40 20,40 10,10</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
return geo:exterior-ring($polygon)
```

### geo:num-interior-ring

**Signatures**

```
geo:num-interior-ring($polygon as element( *)) as xs:integer
```

**Summary**

Returns the number of interior rings in the given polygon $geometry.

**Errors**

- GEO0001: the given element(s) is not recognized as a valid geometry (QName).
- GEO0002: the given element cannot be read by reader for some reason.
- GEO0003: the given element has to be a polygon. Other geometries are not accepted.

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
geo:num-interior-ring(
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>1,1 20,1 20,20 30,20 30,30 1,1</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
    <gml:innerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>2,2 3,2 3,3 2,3 2,2</gml:coordinates>
      </gml:LinearRing>
    </gml:innerBoundaryIs>
    <gml:innerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>10,10 19,10 19,10 10,10</gml:coordinates>
      </gml:LinearRing>
    </gml:innerBoundaryIs>
  </gml:Polygon>
)
geo:interior-ring-n

**Signatures**

geo:interior-ring-n($polygon as element(*)) as element(*)

**Summary**

Returns the outer ring of the given polygon $geometry, as a gml:LineString.

**Errors**

GEO0001: the given element(s) is not recognized as a valid geometry (QName).
GEO0002: the given element cannot be read by reader for some reason.
GEO0003: the given element has to be a polygon. Other geometries are not accepted.
GEO0004: the the input index of geometry is out of range.
GEO0005: the output object cannot be written as an element by writer for some reason.

**Example**

```xml
import module namespace geo = "http://expath.org/ns/geo";
declare namespace gml='http://www.opengis.net/gml';
let $polygon :=
  <gml:Polygon>
    <gml:outerBoundaryIs>
      <gml:LinearRing>
        <gml:coordinates>1,1 20,1 20,20 30,20 30,30 1,30 1,1</gml:coordinates>
      </gml:LinearRing>
    </gml:outerBoundaryIs>
  </gml:Polygon>
return geo:interior-ring-n($polygon, 1)
```

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO0001</td>
<td>Unrecognized Geo type.</td>
</tr>
<tr>
<td>GEO0002</td>
<td>The input GML node cannot be read by GMLreader.</td>
</tr>
<tr>
<td>GEO0003</td>
<td>Input geometry is not an appropriate geometry for this function.</td>
</tr>
<tr>
<td>GEO0004</td>
<td>The input index is out of range.</td>
</tr>
<tr>
<td>GEO0005</td>
<td>The result geometry cannot be written by GMLwriter.</td>
</tr>
<tr>
<td>GEO0006</td>
<td>The input matrix is invalid.</td>
</tr>
</tbody>
</table>

**Changelog**

The module was introduced with Version 7.6.
Chapter 47. Hashing Module

Read this entry online in the BaseX Wiki.

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

| Signatures | hash:md5($value as xs:anyAtomicType) as xs:base64Binary |
| Summary | Computes the MD5 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary. |

Examples

- string(xs:hexBinary(hash:md5("BaseX")))
  
  0D65185C9E296311C0A2200179E479A2.

- string(hash:md5(xs:base64Binary("")))
  
  1B2M2Y8AsqTpgAmY7PhCfg==.

hash:sha1

| Signatures | hash:sha1($value as xs:anyAtomicType) as xs:base64Binary |
| Summary | Computes the SHA-1 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary. |

Examples

- string(xs:hexBinary(hash:sha1("BaseX")))
  
  3AD5958F0F27D5AFFDCA2957560F121D0597A4ED.

- string(hash:sha1(xs:base64Binary("")))
  
  2jmj715rSw0yVb/vIWAYkK/YBwk=.

hash:sha256

| Signatures | hash:sha256($value as xs:anyAtomicType) as xs:base64Binary |
| Summary | Computes the SHA-256 hash of the given $value, which may be of type xs:string, xs:base64Binary, or xs:hexBinary. |

Examples

- string(xs:hexBinary(hash:sha256("BaseX")))
  
  15D570763DEB75D728BB69643392873B835CCC94A2F1E881909DA47662821A3.

- string(hash:sha256(xs:base64Binary("")))
  
  47DEQpj8HBSa+/TImW+5JCeuQeRkm5NmpJWZG3hSuFU=.

hash:hash

| Signatures | hash:hash($value as xs:anyAtomicType, $algorithm as xs:string) as xs:base64Binary |
| Summary | Computes the hash of the given $value, using the specified $algorithm. The specified values may be of type xs:string, xs:base64Binary, or xs:hexBinary. The following three algorithms are supported: MD5, SHA-1, and SHA-256. |
Errors

<table>
<thead>
<tr>
<th>Errors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>algorithm:</td>
<td>the specified hashing 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

<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>

Changelog

Version 9.0

- Updated: error codes updated; errors now use the module namespace

The module was introduced with Version 7.3.
Chapter 48. 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:

  ```xml
  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:

  ```xml
  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:
Higher-Order Functions Module

**hof:until**

```xml
|hof:until{
  function($_) { true() },
  function($_) { error() },
  'OK'
}
```

**hof:scan-left**

**Signatures**

| hof:scan-left($seq as item()*, $start as item()*, $f as function(item()*, item()) as item()*) as item()* |

**Summary**

This function is similar to fn:fold-left, but it returns a list of successive reduced values from the left. It is equivalent to:

```
declare function hof:scan-left($seq, $acc, $f) {
  if(empty($seq)) then $acc else ($acc,
    hof:scan-left(tail($seq), $f($acc, head($seq)), $f)
  );
}
```

**Examples**

- Returns triangular numbers:

  ```xml
  hof:scan-left(1 to 10, 0, function($a, $b) { $a + $b })
  ```

---

**hof:take-while**

**Signatures**

| hof:take-while($seq as item()*, $pred as function(item()) as xs:boolean) as item()* |

**Summary**

The function returns items of $seq as long as the predicate $pred is satisfied. It is equivalent to:

```
declare function hof:take-while($seq, $pred) {
  if(empty($seq) or not($pred(head($seq)))) then () else (head($seq),
    hof:take-while(tail($seq), $pred)
  );
}
```

**Examples**

- Computes at most 100 random integers, but stops if an integer is smaller than 10:

  ```xml
  hof:take-while(
    (1 to 100) ! random:integer(50),
    function($x) { $x >= 10 }
  )
  ```

---

**Sorting**

**hof:top-k-by**

**Signatures**

| hof:top-k-by($seq as item()*, $sort-key as function(item()) as item(), $k as xs:integer) as item()* |

**Summary**

Returns the $k items in $seq that are greatest when sorted by the result of $f applied to the item. The function is a much more efficient implementation of the following scheme:

```
(for $x in $seq
  order by $sort-key($x) descending
  return $x
)```
Higher-Order Functions Module

<table>
<thead>
<tr>
<th>position() &lt;= $k</th>
</tr>
</thead>
</table>

**Examples**

- `hof:top-k-by(1 to 1000, hof:id#1, 5)` returns `1000 999 998 997 996`
- `hof:top-k-by(1 to 1000, function($x) { -$x }, 3)` returns `1 2 3`
- `hof:top-k-by(<x a='1' b='2' c='3'/>/@*, xs:integer#1, 2)/node-name()` returns `c b`

**hof:top-k-with**

**Signatures**

`hof:top-k-with($seq as item()*, $lt as function(item(), item()) as xs:boolean, $k as xs:integer) as item()*`

**Summary**

Returns the $k items in $seq that are greatest when sorted in the order of the less-than predicate $lt. The function is a general version of `hof:top-k-by($seq, $sort-key, $k)`.

**Examples**

- `hof:top-k-with(1 to 1000, function($a, $b) { $a lt $b }, 5)` returns `1000 999 998 997 996`
- `hof:top-k-with(-5 to 5, function($a, $b) { abs($a) gt abs($b) }, 5)` returns `0 1 -1 2 -2`

**IDs**

**hof:id**

**Signatures**

`hof:id($expr as item()*) as item()*`

**Summary**

Returns its argument unchanged. This function isn’t useful on its own, but can be used as argument to other higher-order functions.

**Examples**

- `hof:id(1 to 5)` returns `1 2 3 4 5`
- With higher-order functions:

```xml
let $sort := sort(?, (), hof:id#1)
let $reverse-sort := sort(?, (), function($x) { -$x })
return ($sort((1, 5, 3, 2, 4)),
       '|',
       $reverse-sort((1, 5, 3, 2, 4))
)
```

returns: `1 2 3 4 5 | 5 4 3 2 1`

**hof:const**

**Signatures**

`hof:const($expr as item()*, $ignored as item()*) as item()*`

**Summary**

Returns its first argument unchanged and ignores the second. This function isn’t useful on its own, but can be used as argument to other higher-order functions, e.g. when a function combining two values is expected and one only wants to retain the left one.

**Examples**

- `hof:const(42, 1337)` returns `42`.
- With higher-order functions:

```xml
let $zip-sum := function($f, $seq1, $seq2) {
    sum(for-each-pair($seq1, $seq2, $f))
}
let $sum-all  := $zip-sum(function($a, $b) { $a + $b }, ?, ?)
```

returns: `260`

260
let $sum-left := \text{zip-sum}(\text{hof:const}#2, ?, ?)$
return ($sum-all((1, 1, 1, 1, 1), 1 \text{ to } 5),$ $sum-left((1, 1, 1, 1, 1), 1 \text{ to } 5)$)

• Another use-case: When inserting a key into a map, $f$ decides how to combine the new value with a possibly existing old one. hof:const here means ignoring the old value, so that’s normal insertion.

let $insert-with := \text{function}(f, \text{map}, k, v)$
let $old := \text{map}(k)$
let $new := \text{if}(\text{old}) \text{ then } f(v, \text{old}) \text{ else } v$
return $\text{map:merge}((\text{map}, \text{map:entry}(k, new)))$

let $map := \text{map} \{ \text{\'foo\': 1} \}$
let $add := $insert-with($\text{function}(a, b) \{ a + b \}$, ?, ?, ?)
let $ins := $insert-with(hof:const#2, ?, ?, ?)
return ($add($map, \text{\'foo\'}, 2)(\text{\'foo\'}),$ $ins($map, \text{\'foo\'}, 42)(\text{\'foo\'}))$

returns 3 42

Changelog

Version 8.1
• Added: hof:scan-left, hof:take-while

Version 7.2
• Added: hof:top-k-by, hof:top-k-with
• Removed: hof:iterate

Version 7.0
• module added
Chapter 49. HTML Module

Read this entry online in the BaseX Wiki.

This XQuery Module provides functions for converting HTML to XML. Conversion will only take place if TagSoup is included in the classpath (see HTML Parsing for more details).

Conventions

All functions 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:parser

Signatures

html:parser() as xs:string

Summary

Returns the name of the applied HTML parser (currently: TagSoup). If an empty string is returned, TagSoup was not found in the classpath, and the input will be treated as well-formed XML.

html:parse

Signatures

html:parse($input as xs:anyAtomicType) as document-node()
html:parse($input as xs:anyAtomicType, $options as map(*)?) as document-node()

Summary

Converts the HTML document specified by $input to XML, and returns a document node:

- The input may either be a string or a binary item (xs:hexBinary, xs:base64Binary).
- If the input is passed on in its binary representation, the HTML parser will try to automatically choose the correct encoding.

The $options argument can be used to set TagSoup Options.

Errors

parse: the input cannot be converted to XML.

Examples

Basic Example

The following query converts the specified string to an XML document node.

Query

```xquery
html:parse("<html>"
```

Result

```xml
<html xmlns="http://www.w3.org/1999/xhtml"/>
```

Specifying Options

The next query creates an XML document with namespaces:

Query

```xquery
```
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

```javascript
```

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>
</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 9.0

- Updated: error codes updated; errors now use the module namespace

The module was introduced with Version 7.6.
Chapter 50. HTTP 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 `exerr` 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.

| 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.
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'
```
**HTTP Module**

```html
href='http://www.google.com'>
  <http:header name="User-Agent" value="Opera"/>
</http:request>
})[2]
return try {
  html:parse($binary)
} catch * {
  'Conversion to XML failed: ' || $err:description
}
```

### SVG Data

Content-type ending with +xml, e.g. image/svg+xml.

**Query:**

```html
http:send-request(<http:request method='get'/>, 'http://upload.wikimedia.org/wikipedia/commons/6/6b/Bitmap_VS_SVG.svg')
```

**Result:**

```html
<http:response status="200" message="OK">
  <http:header name="ETag" value="W/11b6d-4ba15ed4"/>
  <http:header name="Age" value="9260"/>
  <http:header name="Date" value="Mon, 14 Mar 2011 19:17:10 GMT"/>
  <http:header name="Content-Length" value="72557"/>
  <http:header name="Content-Type" value="image/svg+xml"/>
  <http:header name="X-Cache-Lookup" value="MISS from knsq22.knams.wikimedia.org:80"/>
  <http:header name="Connection" value="keep-alive"/>
  <http:header name="Server" value="Sun-Java-System-Web-Server/7.0"/>
  <http:header name="X-Cache" value="MISS from knsq22.knams.wikimedia.org"/>
  <http:body media-type="image/svg+xml"/>
</http:response>
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1" width="1063" height="638">
  <defs>
    <linearGradient id="lg0">
      <stop stop-color="#3333ff" offset="0"/>
      <stop stop-color="#3f3fff" stop-opacity="0" offset="1"/>
    </linearGradient>
  ...
</svg>
```

### POST Request

POST request to the BaseX REST Service, specifying a username and password.

**Query:**

```html
let $request :=
  <http:request href='http://localhost:8984/rest'
    method='post' username='admin' password='admin' send-authorization='true'>
    <http:body media-type='application/xml'>
      <query xmlns="http://basex.org/rest">
        <![CDATA[<div>
          for $i in 1 to 3
          return <div>Section {$i }</div>
        </div>]]></query>
  </http:request>
```

266
HTTP Module

```xml
<http:body>
  <http:request>
    return http:send-request($request)
  </http:request>
</http:body>

Result:

  <http:header name="Content-Length" value="135"/>
  <http:header name="Content-Type" value="application/xml"/>
  <http:header name="Server" value="Jetty(6.1.26)"/>
  <http:body media-type="application/xml"/>
</http:response>

<html>
  <div>Section 1</div>
  <div>Section 2</div>
  <div>Section 3</div>
</html>

Errors

<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 51. 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:open: 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:open: 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:open`: 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:open`: 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:open`: 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:open`: The addressed database does not exist or could not be opened.

---

**Changelog**

**Version 8.4**

- Added: `index:token`

**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 52. 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. qxDoc document instances are assigned to the http://www.xqdoc.org/1.0 namespace, which is statically bound to the xqdoc prefix.

## Reflection

### inspect:functions

<table>
<thead>
<tr>
<th>Signatures</th>
<th>inspect:functions() as function(*)</th>
<th>inspect:functions($uri as xs:string) as function(*)</th>
</tr>
</thead>
</table>

<p>| Summary | Returns function items for all user-defined functions (both public and private) that are known in the current query context. If a $uri is specified, the addressed module will be 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. |</p>
<table>
<thead>
<tr>
<th>Examples</th>
<th>Invokes the declared functions and returns their values:</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare %private function local:one() { 12 };</td>
<td></td>
</tr>
<tr>
<td>declare %private function local:two() { 34 };</td>
<td></td>
</tr>
<tr>
<td>for $f in inspect:functions() return $f()</td>
<td>Compile all functions in code.xqm and invokes the function named run:</td>
</tr>
<tr>
<td>let $uri := 'code.xqm'</td>
<td></td>
</tr>
<tr>
<td>let $name := &quot;run&quot;</td>
<td></td>
</tr>
<tr>
<td>for $f in inspect:functions($uri) where local-name-from-QName(function-name($f)) = $name return $f()</td>
<td></td>
</tr>
</tbody>
</table>

### inspect:function-annotations

<table>
<thead>
<tr>
<th>Signatures</th>
<th>inspect:function-annotations($function as function(<em>)?) as map(xs:QName, xs:anyAtomicType</em>)</th>
</tr>
</thead>
</table>

<p>| Summary | Returns the annotations of the specified $function in a map. |</p>
<table>
<thead>
<tr>
<th>Examples</th>
<th>• Returns an empty map:</th>
</tr>
</thead>
<tbody>
<tr>
<td>inspect:function-annotations(true#0)</td>
<td>• Returns a map with a single key Q{<a href="http://www.w3.org/2012/xquery%7Dprivate">http://www.w3.org/2012/xquery}private</a> and an empty sequence as value:</td>
</tr>
<tr>
<td>declare %private function local:f() { 'well hidden' };</td>
<td></td>
</tr>
<tr>
<td>inspect:function-annotations(local:f#0)</td>
<td></td>
</tr>
</tbody>
</table>
**inspect:static-context**

**Signatures**

\[
\text{inspect:static-context($function \text{ as function(*)?}, \ $name \text{ as xs:string}) \text{ as item(*)}}
\]

**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()):

  \[
  \text{inspect:static-context((), 'base-uri')}
  \]

- Returns a map with all namespaces that are statically known in the module of the specified function:

  import module namespace data = 'data.xqm';
  inspect:static-context(data:get#1, 'namespaces')

**Errors**

unknown: The specified component does not exist.

**Documentation**

**inspect:function**

**Signatures**

\[
\text{inspect:function($function \text{ as function(*)) \text{ as element(function)}}
\]

**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:

\[
\begin{xml}
<function name="count" url="http://www.w3.org/2005/xpath-functions"
    external="false">
  <argument type="item()" occurrence="*"/>
\end{xml}
\]
The function...

(:~
  : This function simply returns the specified integer.
  : @param $number number to return
  : @return specified number
:)
declare %private function local:same($number as xs:integer) as xs:integer {
  $number
};

...is represented by inspect:context(local:same#1) as...

<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

<table>
<thead>
<tr>
<th>Signatures</th>
<th>inspect:context() as element(context)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Generates an element that describes all variables and functions in the current query context.</td>
</tr>
<tr>
<td>Examples</td>
<td>Evaluate all user-defined functions with zero arguments in the query context:</td>
</tr>
<tr>
<td></td>
<td>inspect:context()/function ! function-lookup(QName($uri, @name), 0) ! . ()</td>
</tr>
<tr>
<td></td>
<td>Return the names of all private functions in the current context:</td>
</tr>
<tr>
<td></td>
<td>for $f in inspect:context()/function where $f/annotation/@name = 'private' return $f/@name/string()</td>
</tr>
</tbody>
</table>

### inspect:module

<table>
<thead>
<tr>
<th>Signatures</th>
<th>inspect:module($uri as xs:string) as element(module)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>An example is shown below.</td>
</tr>
</tbody>
</table>

### inspect:xqdoc

<table>
<thead>
<tr>
<th>Signatures</th>
<th>inspect:xqdoc($uri as xs:string) as element(xqdoc:xqdoc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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</td>
</tr>
</tbody>
</table>
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.

Examples

This is the sample.xqm library module:

(:~
 : This module provides some sample functions to demonstrate
 : the features of the Inspection Module.
 : @author BaseX Team
 : @see http://docs.basex.org/wiki/XQDoc_Module
 : @version 1.0
 :)
module namespace samples = 'http://basex.org/modules/samples';

(:~ This is a sample string. :) 
declare variable $samples:test-string as xs:string := 'this is a string';

(:~
 : This function simply returns the specified integer.
 : @param $number number to return
 : @return specified number
 :)
declare %private function samples:same($number as xs:integer) as xs:integer {
    $number
};

If inspect:module('sample.xqm') is run, the following output will be generated:

<module prefix="samples" uri="http://basex.org/modules/samples">
   <description>This module provides some sample functions to demonstrate
   the features of the Inspection Module.</description>
   <author>BaseX Team</author>
   <see>http://docs.basex.org/wiki/XQDoc_Module</see>
   <version>1.0</version>
   <variable name="samples:test-string" uri="http://basex.org/modules/samples"
   type="xs:string" 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" url="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: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:name>samples:same</xqdoc:name>
      <xqdoc:signature>declare %private function samples:same($number as xs:integer) as xs:integer</xqdoc:signature>
      <xqdoc:parameters>
        <xqdoc:parameter>
          <xqdoc:name>number</xqdoc:name>
          <xqdoc:type>xs:integer</xqdoc:type>
        </xqdoc:parameter>
      </xqdoc:parameters>
      <xqdoc:return>
        <xqdoc:type>xs:integer</xqdoc:type>
      </xqdoc:return>
    </xqdoc:function>
  </xqdoc:functions>
</xqdoc:xqdoc>
```

## Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unknown</td>
<td>The specified component does not exist.</td>
</tr>
</tbody>
</table>
Changelog

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 53. Jobs Module

Read this entry online in the BaseX Wiki.

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/jobs` namespace, which is statically bound to the `jobs` prefix. Errors will be bound to the same prefix.

Services

Jobs can additionally be registered as persistent services. An additional service option has been added to the options list of `jobs:eval` and `jobs:invoke`:

```xquery
jobs:eval('db:drop("tmp")', (), map { 'id':'cleanup', 'start':'01:00:00', 'interval':'P1D', 'service': true() }),
```

```xquery
jobs:services(),
```

```xquery
jobs:stop('cleanup', map { 'service': true() })
```

Some more notes:

- All registered jobs 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.
- Job services can be updated: If a new job is registered, and if there is already a job with the same id, the old entry will be replaced.
- The job definitions are stored in a `jobs.xml` file in the database directory. It can also be edited manually.

Basic Functions

**jobs:current**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>jobs:current() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the id of the current job.</td>
</tr>
</tbody>
</table>

**jobs:list**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>jobs:list() as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the ids of all jobs that are currently registered. The list includes scheduled, queued, running, stopped, and finished jobs with cached results.</td>
</tr>
<tr>
<td>Examples</td>
<td>jobs:list() returns the same job id as jobs:current if no other job is registered.</td>
</tr>
</tbody>
</table>
**jobs:list-details**

*Updated with Version 9.1: registration time added*

**Signatures**

\[
\begin{array}{l}
\text{jobs:list-details()} \quad \text{as element(job)*}
\end{array}
\]

\[
\begin{array}{l}
\text{jobs:list-details($id \text{ as xs:string}) as element(job)*}
\end{array}
\]

**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 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
- **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**

\[
\begin{array}{l}
\text{jobs:list-details()} \text{ returns information on the currently running job and possibly others:}
\end{array}
\]

\[
\begin{array}{l}
\text{<job id="job1" type="XQuery" state="running" user="admin" duration="PT0.001S">}
\text{XQUERY \text{jobs:list-details()}}
\text{</job>}
\end{array}
\]

**jobs:finished**

**Signatures**

\[
\begin{array}{l}
\text{jobs:finished($id \text{ as xs:string}) as xs:boolean}
\end{array}
\]

**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).

**jobs:services**

**Signatures**

\[
\begin{array}{l}
\text{jobs:services()} \quad \text{as element(job)*}
\end{array}
\]

**Summary**

Returns a list of all jobs that have been persistently registered as Services.

**Errors**

\[
\begin{array}{l}
\text{services: Registered services cannot be parsed.}
\end{array}
\]

**Execution**

Asynchronous query execution is recommendable if a client does not, or cannot, wait until a request is fully processed. This is e. g. the case with web browsers, which will usually cancel a request after a specific timeout. In such cases, you can use asynchronous execution to trigger another server-side process, which will start the time-consuming process, and fetch the result later on as soon as it is available.
**jobs:eval**

**Signatures**

```xquery
jobs:eval($query as xs:string) as xs:string
jobs:eval($query as xs:string, $bindings as map(*)?) as xs:string
jobs:eval($query as xs:string, $bindings as map(*)?, $options as map(*)?) as xs:string
```

**Summary**

Schedules the evaluation of the supplied $query and returns a query id. The query will be queued, and the result will optionally be cached. Queries can be updating. Variables and context items 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 jobs:result, or until CACHETIMEOUT is exceeded.
  - If the query raises an error, it will be cached and returned instead.

- **start**: a dayTimeDuration, time or dateTime can be specified to delay the execution of the query:
  - If a dayTimeDuration is specified, the query will be queued after the specified duration has passed. Examples for valid values are: P1D (1 day), PT5M (5 minutes), PT0.1S (100 ms). An error will be raised if a negative value is specified.
  - If a time is specified, the query will be executed at this time of the day. Examples for valid times are: 02:00:00 (2am local time), 12:00:00Z (noon, UTC). If the time lies in the past, the query will be executed the next day.
  - If a dateTime is specified, the query will be executed at this date. Examples for valid values are: 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.

- **interval**: a dayTimeDuration string can be specified to execute the query periodically. An error is raised if the specified interval is less than one second (PT1S). If the next scheduled call is due, and if a query with the same id is still running, it will be skipped.

- **end**: scheduling can be stopped after a given time or duration. The string format is the same as for start. An error is raised if the resulting end time is smaller than the start time.

- **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.

- **service**: additionally registers the job as service.

**Errors**

- **overflow**: Query execution is rejected, because too many jobs are queued or being executed. CACHETIMEOUT can be decreased if the default setting is too restrictive.
- **range**: A specified time or duration is out of range.
- **id**: The specified id is invalid or has already been assigned.
- **options**: The specified options are conflicting.

**Examples**

- Cache query result. The returned id can be used to pick up the result with jobs:result:

  ```xquery
  jobs:eval("1+3", (), map { 'cache': true() })
  ```

- A happy birthday mail will be sent at the given date:

  ```xquery
  jobs:eval("import module namespace mail='mail'; mail:send('Happy birthday!')")
  ```
• The following 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:

```xml
declare %rest:POST("{$query}" ) %rest:path('/start-scheduling' ) function
local:start($query) {
   jobs:eval($query, (), map { 'start': '02:00:00', 'interval': 'P1D' })
};
```

```xml
declare %rest:path('/stop-scheduling/{$id}') function local:stop($id) {
   jobs:stop($id)
};
```

• 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
jobs:eval("prof:sleep(1500)", (), map { 'interval': 'PT1S', 'end': 'PT10S' })
```

• The following expression, if stored as a file, calls and evaluates itself every 5 seconds:

```xml
jobs:eval(
   file:read-text(static-base-uri()),
   map { },
   map { 'start': 'PT5S' }
)
```

### jobs:invoke

**Signatures**

| jobs:invoke($uri as xs:string) as xs:string |
| jobs:invoke($uri as xs:string, $bindings as map(*)) as xs:string |
| jobs:invoke($uri as xs:string, $bindings as map(*), $options as map(*)) as xs:string |

**Summary**

Schedules the evaluation of the XQuery expression located at $uri and returns a query id. For further details, see jobs:eval.

**Errors**

overflow: Query execution is rejected, because too many jobs are queued or being executed.
CACHETIMEOUT can be decreased if the default setting is too restrictive.
range: A specified time or duration is out of range.
id: The specified id is invalid or has already been assigned.
options: The specified options are conflicting.

**Examples**

Run XQuery expression that may perform some cleanups:

```xml
jobs:invoke("cleanup.xq", (), ())
```

### jobs:result

**Signatures**

| jobs:result($id as xs:string) as item(*) |

**Summary**

Returns the cached result of a job with the specified job $id:

• Results can only be retrieved once. After retrieval, the cached result will be dropped.

• If the original job has raised an error, the cached error will be raised instead.

**Errors**

running: the job is still running.
unknown: the supplied id is unknown: The id is unknown, or the result has already been retrieved.

**Examples**

• The following RESTXQ function will either return the result of a previously started job or raise an error:
• The following query demonstrates how the results of an asynchronously executed query can be returned within the same query:

```xml
let $query := jobs:eval('(1 to 1000000)[. = 1]', map{}, map{ 'cache': true() })
return (jobs:wait($query), jobs:result($query))
```

Please note that queries of this kind can cause deadlocks. For example, if both the original query and the query to be executed asynchronously perform updates on the same database, the second query would only be run after the first one has been executed, and the first query will wait forever. This is why you should avoid this pattern in practice and resort to `xquery:fork-join` if you want to do things in parallel.

---

**jobs:stop**

**Signatures**

`jobs:stop($id as xs:string) as empty-sequence()`

**Summary**

Triggers the cancellation of a job with the specified `$id`, drops the cached result of a query, or cancels a scheduled job. Unknown ids are ignored. All jobs are gracefully stopped; it is up to the process to decide when it is safe to shut down. The following `$options` can be supplied:

- `service`: additionally removes the job from the `job services` list.

**Examples**

```
jobs:list()[. != jobs:current()] ! jobs:stop(.)
```

**jobs:wait**

**Signatures**

`jobs: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 cannot be addressed.

---

**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>
unknown

The supplied query id is unknown or not available anymore.

Changelog

Version 9.1

• Updated: jobs:list-details: registration time added.

Version 9.0

• Added: jobs:invoke, Services

Version 8.6

• Updated: jobs:eval: id option added.

The module was introduced with Version 8.5.
Chapter 54. 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 rewritten to an element name:
  • Empty names are represented by a single underscore (_). Existing underscores are rewritten to two underscores (____), and characters that are not valid in element names are rewritten to an underscore and the character’s four-digit Unicode.
  • If the lax option is set to true, invalid characters are simply replaced with underscores or (when invalid as first character of an element name) prefixed with an underscore. The resulting names are better readable, but cannot always be converted back to their original form.

• Array entries are also represented via elements. _ is used as element name.

• Object and array values are stored in text nodes.

• The types of values are represented via type attributes:
  • The existing types are string, number, boolean, null, object, and array.
  • As most values are strings, the string type is by default omitted.

Attributes

The attributes format is lossless, too. The transformation based on the following rules:

• The resulting document has a <json> root node.

• Object pairs are represented via <pair> elements. The name of a pair is stored in a name attribute.

• Array entries are represented via <item> elements.

• Object and array values are stored in text nodes.

• The types of values are represented via type attributes:
• The existing types are `string`, `number`, `boolean`, `null`, `object`, and `array`.

• As most values are strings, the `string` type is by default omitted.

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 <code>type</code> 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 (<code>numbers</code>, <code>booleans</code>, <code>nulls</code>, <code>objects</code> and <code>arrays</code>), 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, <code>\n</code> becomes a single <code>\n</code> character, while <code>\u20AC</code> 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 <code>no</code> if strings</td>
<td>yes, no</td>
<td>yes</td>
<td>serialize</td>
</tr>
</tbody>
</table>
are already in escaped form and no further escaping is permitted.

<table>
<thead>
<tr>
<th>indent</th>
<th>Indicates if whitespace should be added to the output with the aim of improving human legibility. If the parameter is set as in the query prolog, it overrides the indent serialization parameter.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes, no</td>
</tr>
</tbody>
</table>

**Functions**

**json:parse**

*Updated with Version 9.1:* support for empty sequence.

**Signatures**

<table>
<thead>
<tr>
<th>json:parse($string as xs:string?) as item()? json:parse($string as xs:string?, $options as map(*)?) as item()?</th>
</tr>
</thead>
</table>

**Summary**

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.

**Errors**

parse: the specified input cannot be parsed as JSON document. $options: the specified options are conflicting.

**json:serialize**

**Signatures**

<table>
<thead>
<tr>
<th>json:serialize($input as item())? json:serialize($input as item(), $options as map(*))? as xs:string</th>
</tr>
</thead>
</table>

**Summary**

Serializes the specified $input as JSON, using the specified $options, and returns the result as string:

- The input is expected to conform to the results that are created by json:parse().
- Non-conforming items will be serialized as specified in the json output method of the official recommendation.

Values can also be serialized as JSON with the standard Serialization feature of XQuery:

- The parameter method needs to be set to json, and
- the options presented in this article need to be assigned to the json parameter.

**Errors**

serialize: the specified node cannot be serialized as JSON document.

**Examples**

**BaseX Format**

**Example 1: Adds all JSON documents in a directory to a database**

**Query:**

```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:**

```xml
284```
**Example 3: Converts a JSON string with simple objects and arrays**

**Query:**

```plaintext
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:**

```plaintext
<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:**

```plaintext
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"},
```
Example 3: Converts a document with nested elements and attributes to JsonML

Query:

```
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:

```
["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:

```
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:

```
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 9.1

• 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 55. 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 requested. 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:
  - db:retrieve
  - fetch:binary
  - file:read-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-bytes, convert:binary-to-string
- Database Module: db:store
- File Module: file:write-binary, file:write-text (if no encoding is specified)
- 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:

```xquery
```

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**

*Updated with Version 9.1:* $lazy$ argument added; support for sequences.
Lazy Module

### Signatures

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
</table>
| lazy:cache($items as item()* as item())* lazy:cache($items as item()* as item()), $lazy as xs:boolean) as item()* | 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

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy:is-lazy($item as item()) as xs:boolean</td>
<td>Checks whether the specified $item is lazy.</td>
</tr>
</tbody>
</table>

### lazy:is-cached

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy:is-cached($item as item()) as xs:boolean</td>
<td>Checks whether the contents of the specified $item are cached. The function will always return true for non-lazy items.</td>
</tr>
</tbody>
</table>

### 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 56. Map Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for manipulating maps. Maps have been introduced with XQuery 3.1 and are described in detail in the XQuery Functions and Operators 3.1 specification.

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")
)
```


Map Module

Unlike the `map { ... }` expression, this technique can be used to construct a map with a variable number of entries, for example:

```xml
map:merge(for $b in //book return map:entry($b/isbn, $b))
```

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map:entry(&quot;M&quot;, &quot;Monday&quot;)</code> creates map { &quot;M&quot;: &quot;Monday&quot; }</td>
<td></td>
</tr>
</tbody>
</table>

**map:find**

**Signatures**

```xml
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**

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map:find(map { 1:2 }, 1)</code> returns [ 2 ]</td>
<td></td>
</tr>
<tr>
<td><code>map:find(map { 1: map { 2: map { 3: 4 } } }, 3)</code> returns [ 4 ]</td>
<td></td>
</tr>
<tr>
<td><code>map:find((1, 'b', true#0), 1)</code> returns an empty array</td>
<td></td>
</tr>
</tbody>
</table>

**map:for-each**

**Signatures**

```xml
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):

```xml
map:for-each(
  map { 1: 2, 3: 4 },
  function($key, $value) { $key + $value }
)
```

**map:get**

**Signatures**

```xml
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, $K)`. Similarly, the expression `get(get($map, 'employee'), 'name'), 'first'` can be written as `map('employee')('name')('first')`.

**Examples**

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>map:get($week, 4)</code> returns &quot;Thu&quot;.</td>
<td></td>
</tr>
<tr>
<td><code>map:get($week, 9)</code> returns () (When the key is not present, the function returns an empty sequence.)</td>
<td></td>
</tr>
</tbody>
</table>
Map Module

- `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`.

**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:remove($week, 4)` creates map 

- `map:remove($week, 23)` creates map 

---

### `map:size`

**Signatures**

`map:size($map as map(*)) as xs:integer`

**Summary**

Returns 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 57. Math Module

The math XQuery Module defines functions to perform mathematical operations, such as \( \text{pi} \), \( \text{asin} \) and \( \text{acos} \). 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**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:pi() as xs:double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the ( \text{xs:double} ) value of the mathematical constant ( \pi ) whose lexical representation is 3.141592653589793.</td>
</tr>
<tr>
<td>Examples</td>
<td>* 2*math:pi() returns 6.283185307179586e0.</td>
</tr>
<tr>
<td></td>
<td>* 60 * (math:pi() div 180) converts an angle of 60 degrees to radians.</td>
</tr>
</tbody>
</table>

**math:sqrt**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:sqrt($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the square root of ( $arg ) If ( $arg ) is the empty sequence, the empty sequence is returned. Otherwise the result is the ( \text{xs:double} ) value of the mathematical square root of ( $arg ).</td>
</tr>
</tbody>
</table>

**math:sin**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:sin($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the sine of the ( $arg ), expressed in radians. If ( $arg ) is the empty sequence, the empty sequence is returned. Otherwise the result is the sine of ( $arg ), treated as an angle in radians.</td>
</tr>
</tbody>
</table>

**math:cos**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:cos($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the cosine of ( $arg ), expressed in radians. If ( $arg ) is the empty sequence, the empty sequence is returned. Otherwise the result is the cosine of ( $arg ), treated as an angle in radians.</td>
</tr>
</tbody>
</table>

**math:tan**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:tan($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the tangent of ( $arg ), expressed in radians. If ( $arg ) is the empty sequence, the empty sequence is returned. Otherwise the result is the tangent of ( $arg ), treated as an angle in radians.</td>
</tr>
</tbody>
</table>

**math:asin**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>math:asin($arg as xs:double?) as xs:double?</th>
</tr>
</thead>
</table>
Math Module

**Summary**

Returns the arc sine of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the arc sine of $arg$, returned as an angle in radians in the range $-\pi/2$ to $+\pi/2$.

**math:acos**

**Signatures**

```
math:acos($arg as xs:double?) as xs:double?
```

**Summary**

Returns the arc cosine of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the arc cosine of $arg$, returned as an angle in radians in the range $0$ to $+\pi$.

**math:atan**

**Signatures**

```
math:atan($arg as xs:double?) as xs:double?
```

**Summary**

Returns the arc tangent of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the arc tangent of $arg$, returned as an angle in radians in the range $-\pi/2$ to $+\pi/2$.

**math:atan2**

**Signatures**

```
math:atan2($arg1 as xs:double?, $arg2 as xs:double) as xs:double?
```

**Summary**

Returns the arc tangent of $arg1$ divided by $arg2$, the result being in the range $-\pi/2$ to $+\pi/2$ radians. If $arg1$ is the empty sequence, the empty sequence is returned. Otherwise the result is the arc tangent of $arg1$ divided by $arg2$, returned as an angle in radians in the range $-\pi$ to $+\pi$.

**math:pow**

**Signatures**

```
math:pow($arg1 as xs:double?, $arg2 as xs:double) as xs:double?
```

**Summary**

Returns $arg1$ raised to the power of $arg2$. If $arg1$ is the empty sequence, the empty sequence is returned. Otherwise the result is the $arg1$ raised to the power of $arg2$.

**Examples**

• `math:pow(2, 3)` returns 8.

**math:exp**

**Signatures**

```
math:exp($arg as xs:double?) as xs:double?
```

**Summary**

Returns $e$ raised to the power of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the value of $e$ raised to the power of $arg$.

**Examples**

• `math:exp(1)` returns $e$.

**math:log**

**Signatures**

```
math:log($arg as xs:double?) as xs:double?
```

**Summary**

Returns the natural logarithm of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the natural logarithm (base $e$) of $arg$.

**Examples**

• `math:log(math:e())` returns 1.

**math:log10**

**Signatures**

```
math:log10($arg as xs:double?) as xs:double?
```

**Summary**

Returns the base 10 logarithm of $arg$. If $arg$ is the empty sequence, the empty sequence is returned. Otherwise the result is the base 10 logarithm of $arg$.

**Examples**

• `math:log(100)` returns 2.
Additional Functions

**math:e**

Signatures  
math:e() as xs:double

Summary  
Returns the xs:double value of the mathematical constant e whose lexical representation is 2.718281828459045.

Examples  
• 5*math:e() returns 13.591409142295225.

**math:sinh**

Signatures  
math:sinh($arg as xs:double?) as xs:double?

Summary  
Returns the hyperbolic sine of $arg. If $arg is the empty sequence, the empty sequence is returned. Otherwise the result is the hyperbolic sine of $arg.

Examples  
• math:sinh(0) returns 0.

**math:cosh**

Signatures  
math:cosh($arg as xs:double?) as xs:double?

Summary  
Returns the hyperbolic cosine of $arg. If $arg is the empty sequence, the empty sequence is returned. Otherwise the result is the hyperbolic cosine of $arg.

Examples  
• math:cosh(0) returns 1.

**math:tanh**

Signatures  
math:tanh($arg as xs:double?) as xs:double?

Summary  
Returns the hyperbolic tangent of $arg. If $arg is the empty sequence, the empty sequence is returned. Otherwise the result is the hyperbolic tangent of $arg.

Examples  
• math:tanh(100) returns 1.

**math:crc32**

Updated with Version 9.1: support for empty sequence.

Signatures  
math:crc32($string as xs:string?) as xs:hexBinary?

Summary  
Calculates the CRC32 check sum of the given $string. If an empty sequence is supplied, the empty sequence is returned.

Examples  
• math:crc32("") returns '00000000'.
  
  • 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 Random Module.

Version 7.3

• Added: math:crc32 and math:uuid have been adopted from the obsolete Utility Module.
Chapter 58. Output Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for simplifying formatted data output.

Conventions

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

Functions

out:cr

Signatures  out:cr() as xs:string

Summary  Returns a single carriage return character (\&#13;).

out:nl

Signatures  out:nl() as xs:string

Summary  Returns a single newline character (\&#10;).

out:tab

Signatures  out:tab() as xs:string

Summary  Returns a single tabulator character (\&#9;).

out:format

Signatures  out: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  • out:format("%b", true()) returns true.
  • out:format("%06d", 256) returns 000256.
  • out:format("%e", 1234.5678) returns 1.234568e+03.

Errors

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>The specified format is not valid.</td>
</tr>
</tbody>
</table>

Changelog

Version 9.0

• Added: out:cr

• Updated: error codes updated; errors now use the module namespace
Introduced with Version 7.3. Functions have been adopted from the obsolete Utility Module.
Chapter 59. 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**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proc:system($cmd as xs:string) as xs:string</code></td>
<td>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 $options parameter contains process options:</td>
</tr>
<tr>
<td><code>proc:system($cmd as xs:string, $args as xs:string*) as xs:string</code></td>
<td></td>
</tr>
<tr>
<td><code>proc:system($cmd as xs:string, $args as xs:string*, $options as map(xs:string, xs:string)) as xs:string</code></td>
<td></td>
</tr>
</tbody>
</table>

**Summary**

- **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):

  ```xquery```
  ```
  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, $args as xs:string*) as element(result)`
- `proc:execute($cmd as xs:string, $args as xs:string*, $options as map(xs:string, xs:string)) as element(result)`

**Summary**

Executes the specified command in a separate process and returns the result as element:

- `$cmd` is the name of the command, and arguments to the command may be specified via `$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

**Signatures**

- `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)`

**Summary**

Executes the specified command and ignores the result. `$cmd` is the name of the command, and arguments to the command may be specified via `$args`. The same `$options` are allowed as for `proc:system` (but the encoding will be ignored).

**Errors**

- **encoding**: the specified encoding does not exist or is not supported.

**Examples**

- `proc:fork('sleep', '5')`: sleep for 5 seconds (no one should notice).

### proc:property

**Signatures**

- `proc:property($name as xs:string) as xs:string?`

**Summary**

Returns the system property, specified by `$name`, or a context parameter of the `web.xml` 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 `fn:environment-variable`.

**Examples**

- `proc:property('java.class.path')` returns the full user class path.
- `map:merge(proc:property-names() ! map:entry(.,, proc:property(.)))` returns a map with all system properties.
**proc:property-names**

**Signatures**

\[
\text{proc:property-names()} \ \text{as} \ \text{xs:string}* 
\]

**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 60. 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:

- **memory**: Include memory consumption in result (unit: bytes; default: true).
- **time**: Include execution time in result (unit: milliseconds; default: true).
- **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:xml is used, as fn:doc may already be evaluated at compilation time):

  prof:track(fetch:xml('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**

\[
\text{prof:time($expr \text{ as item()}) \text{ as item()}* \text{prof:time($expr \text{ as item()},}$} \\
\text{$label \text{ as xs:string)} \text{ as item()*)}
\]

**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**

\[
\text{prof:memory($expr \text{ as item()}) \text{ as item()}* \text{prof:memory($expr \text{ as item()},}$} \\
\text{$label \text{ as xs:string)} \text{ as item()*)}
\]

**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**

\[\text{prof:current-ms() as xs:integer}\]

**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**

\[\text{prof:current-ns() as xs:integer}\]

**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:

\[
\begin{align*}
\text{let } &\$ns1 := \text{prof:current-ns} \\
\text{return } &\left( \begin{array}{l}
\text{(: process to measure :)} \\
\text{(1 to 1000000)[. = 0]}, \\
\text{let } &\$ns2 := \text{prof:current-ns} \\
\text{let } &\$ms := (((\$ns2 - \$ns1) \text{idiv 10000}) \text{idiv 100}) \\
\text{return } &\$ms || \text{ ' ms'}
\end{array}\right)
\end{align*}
\]

### Debugging Functions

#### prof:dump

**Signatures**

\[
\text{prof:dump($expr \text{ as item()}) \text{ as empty-sequence()prof:dump($expr \text{ as item()},$label \text{ as xs:string}) as empty-sequence()}
\]
### Profiling Module

**Summary**
Dumps a serialized representation of `$expr` to STDERR, optionally prefixed with `$label`, and returns an empty sequence. If the GUI is used, the dumped result is shown in the Info View.

**Properties**
In contrast to `fn:trace()`, the consumed expression will not be passed on.

#### `prof:variables`

**Signatures**
`prof:variables() as empty-sequence()

**Summary**
Prints a list of all current local and global variable assignments to standard error or, if the GUI is used, to the Info View. As every query is optimized before being evaluated, not all of the original variables may be visible in the output. Moreover, many variables of function calls will disappear because functions are inlined. Function inlining can be turned off by setting the `LINELIMIT` option to 0.

**Properties**
The function is *non-deterministic*: evaluation order will be preserved by the compiler.

**Examples**
- `for $x in 1 to 2 return prof:variables()` will dump the values of `$x` to standard error.

#### `prof:type`

**Signatures**
`prof:type($expr as item()*) as item()*

**Summary**
Similar to `fn:trace($expr, $msg)`, but instead of a user-defined message, it emits the compile-time type and estimated result size of its argument.

#### Helper Functions

#### `prof:void`

**Signatures**
`prof:void($value as item()*) as empty-sequence()

**Summary**
Swallows all items of the specified `$value` and returns an empty sequence. This function is helpful if some code needs to be evaluated and if the actual result is irrelevant.

**Properties**
The function is *non-deterministic*: evaluation order will be preserved by the compiler.

**Examples**
- `prof:void(fetch:binary('http://my.rest.service'))` performs an HTTP request and ignores the result.

#### `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`.

### Changelog

Version 9.0

- Added: `prof:track`
- Updated: renamed `prof:mem` to `prof:memory`, `prof:time`: `$cache` argument removed
Profiling Module

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 61. Random Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for computing random values. All functions except for random:seeded-double and random:seeded-integer are non-deterministic, i.e., they return different values for each call.

Conventions

All functions 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

<table>
<thead>
<tr>
<th>Signatures</th>
<th>random:double() as xs:double</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns a double value between 0.0 (inclusive) and 1.0 (exclusive).</td>
</tr>
</tbody>
</table>

random:integer

<table>
<thead>
<tr>
<th>Signatures</th>
<th>random:integer() as xs:integer random:integer($max as xs:integer) as xs:integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns an integer value, either in the whole integer range or between 0 (inclusive) and the given maximum (exclusive)</td>
</tr>
<tr>
<td>Errors</td>
<td>bounds: the maximum value is out of bounds.</td>
</tr>
</tbody>
</table>

random:seeded-double

<table>
<thead>
<tr>
<th>Signatures</th>
<th>random:seeded-double($seed as xs:integer, $num as xs:integer) as xs:double*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>

random:seeded-integer

<table>
<thead>
<tr>
<th>Signatures</th>
<th>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*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>bounds: the maximum value is out of bounds. negative: the number of values to be returned is negative.</td>
</tr>
</tbody>
</table>

random:gaussian

<table>
<thead>
<tr>
<th>Signatures</th>
<th>random:gaussian($num as xs:integer) as xs:double*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 62. Repository Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for installing, listing and deleting modules contained in the Repository.

Conventions

All functions 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($path as xs:string) as empty-sequence() |
| Summary | Installs a package or replaces an existing package. The parameter $path indicates the path to the package. |
| Errors | 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($pkg as xs:string) as empty-sequence() |
| Summary | Deletes a package. The parameter $pkg indicates 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 63. Request Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for retrieving information on an HTTP request that has triggered the query. It is mainly useful in the context of Web Applications.

The module is related to Adam Retter’s EXQuery Request Module draft.

Conventions

- The 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. The module must be imported in the query prolog:

  ```
  import module namespace request = "http://exquery.org/ns/request";
  ...
  ```

- In this document, the namespace is 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
```

<table>
<thead>
<tr>
<th>scheme</th>
<th>hostname</th>
<th>port</th>
<th>path</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo</td>
<td>example.com</td>
<td>8042</td>
<td>/over/there</td>
<td>?name=ferret</td>
</tr>
</tbody>
</table>

General Functions

**request:method**

**Signatures**

request:method() as xs:string

**Summary**

Returns the Method of the HTTP request.

**request:attribute**

**Signatures**

request:attribute($name as xs:string) as xs:string

**Summary**

Returns the value of an attribute of the HTTP request. If the attribute does not exist, an empty sequence is returned.

**Example**

- request:attribute("javax.servlet.error.request_uri") returns the original URI of a caught error.

- request:attribute("javax.servlet.error.message") returns the error message of a caught error.

URI Functions

**request:scheme**

**Signatures**

request:scheme() as xs:string
### request:hostname

**Signatures**
request:hostname() as xs:string

**Summary**
Returns the Hostname component of the URI of an HTTP request.

**Example**
For the example given in the introduction, this function would return `example.com`.

### request:port

**Signatures**
request:port() as xs:integer

**Summary**
Returns the Port component of the URI of an HTTP request, or a default port if it has not been explicitly specified in the URI.

**Example**
For the example given in the introduction, this function would return `8042`.

### request:path

**Signatures**
request:path() as xs:string

**Summary**
Returns the Path component of the URI of an HTTP request.

**Example**
For the example given in the introduction, this function would return `/over/there`.

### request:query

**Signatures**
request:query() as xs:string?

**Summary**
Returns the Query component of the URI of an HTTP request. If no query has been specified, an empty sequence is returned.

**Example**
For the example given in the introduction, this function would return `name=ferret`.

### request:uri

**Signatures**
request:uri() as xs:anyURI

**Summary**
Returns the complete URI of an HTTP request as it has been specified by the client.

**Example**
For the example given in the introduction, this method would return `foo://example.com:8042/over/there?name=ferret`.

### request:context-path

**Signatures**
request:context-path() as xs:string

**Summary**
Returns the context of the request. For servlets in the default (root) context, this method returns an empty string.

### Connection Functions

### request:address

**Signatures**
request:address() as xs:string

**Summary**
Returns the IP address of the server.
### Request Module

**request:remote-hostname**

**Signatures**  
`request:remote-hostname() as xs:string`

**Summary**  
Returns the fully qualified hostname of the client that sent the request.

**request:remote-address**

**Signatures**  
`request:remote-address() as xs:string`

**Summary**  
Returns the IP address of the client that sent the request.

**request:remote-port**

**Signatures**  
`request:remote-port() as xs:string`

**Summary**  
Returns the TCP port of the client socket that triggered the request.

### Parameter Functions

**request:parameter-names**

**Signatures**  
`request:parameter-names() as xs:string*`

**Summary**  
Returns the names of all query and form field parameters available from the HTTP request. With RESTXQ, this function can be used to access parameters that have not been statically bound by `%rest:query-param`.

**Example**  
For the example given in the introduction, this function would return `name`.

**request:parameter**

**Signatures**  
`request:parameter($name as xs:string) as xs:string*`  
`request:parameter($name as xs:string, $default as xs:string) as xs:string*`

**Summary**  
Returns the value of the named query or form field parameter in an HTTP request. If the parameter does not exist, an empty sequence or the optionally specified default value is returned instead. If both query and form field parameters with the same name exist, the form field values will be attached to the query values.

**Example**  
For the example given in the introduction, the function call `request:parameter('name')` would return `ferret`.

### Header Functions

**request:header-names**

**Signatures**  
`request:header-names() as xs:string*`

**Summary**  
Returns the names of all headers available from the HTTP request. If RESTXQ is used, this function can be used to access headers that have not been statically bound by `%rest:header-param`.

**request:header**

**Signatures**  
`request:header($name as xs:string) as xs:string?`  
`request:header($name as xs:string, $default as xs:string) as xs:string`

**Summary**  
Returns the value of the named header in an HTTP request. If the header does not exist, an empty sequence or the optionally specified default value is returned instead.
Cookie Functions

request:cookie-names

Signatures
request:cookie-names() as xs:string*

Summary
Returns the names of all cookies in the HTTP headers available from the HTTP request. If RESTXQ is used, this function can be used to access cookies that have not been statically bound by %rest:cookie-param.

request:cookie

Signatures
request:cookie($name as xs:string) as xs:string*
request:cookie($name as xs:string, $default as xs:string) as xs:string

Summary
Returns the value of the named Cookie in an HTTP request. If there is no such cookie, an empty sequence or the optionally specified default value is returned instead.

Changelog

Version 7.9

• Updated: The returned values of request:parameter-names, request:parameter now also include form field parameters.

Version 7.8

• Added: request:context-path

Version 7.7

• Added: request:attribute

This module was introduced with Version 7.5.
Chapter 64. RESTXQ Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains helper functions for the RESTXQ API, some of which are defined in the RESTXQ Draft.

Conventions

- The 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. The module must be imported in the query prolog:

  ```
  import module namespace rest = "http://exquery.org/ns/restxq";
  ...
  ```

- In this document, the namespace is 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()
```

**Summary**

Initializes the RESTXQ module cache. This function should be called after RESTXQ modules have been replaced while the web server is running, and if PARSERESTXQ is not set to 0.

Changelog

Version 8.6

- Added: rest:init
This module was introduced with Version 7.7.
Chapter 65. Session Module

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. The module must be imported in the query prolog:

```xquery
import module namespace session = "http://basex.org/modules/session";
...
```

• In this document, the namespace is 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>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>

```xquery
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>
</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>
</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 variables 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>

```xquery
import module namespace session = "http://basex.org/modules/session";
```
### session:get

**Signatures**

```xml
session:get($name as xs:string) 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.

**Errors**

get: The value of an attribute could not be retrieved.

**Examples**

Running the server-side XQuery file `get.xq` via `http://localhost:8984/get.xq?key=user`:

```xml
import module namespace session = "http://basex.org/modules/session";
'Value of ' || $key || ': ' || session:get($key)
```

### session:set

**Signatures**

```xml
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**

set: The supplied value cannot be materialized.

**Examples**

Running the server-side XQuery file `set.xq` via `http://localhost:8984/set.xq?key=user&value=john`:

```xml
import module namespace session = "http://basex.org/modules/session";
session:set($key, $value), 'Variable was set.'
```

### session:delete

**Signatures**

```xml
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`:

```xml
import module namespace session = "http://basex.org/modules/session";
session:delete($key), 'Variable was deleted.'
```

### session:close

**Signatures**

```xml
session:close() as empty-sequence()
```

**Summary**

Unregisters a session and all data associated with it.

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>The stored attribute is no XQuery value.</td>
</tr>
<tr>
<td>set</td>
<td>The supplied value cannot be materialized.</td>
</tr>
</tbody>
</table>

### Changelog

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 66. 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. The module must be imported in the query prolog:

```xml
import module namespace sessions = "http://basex.org/modules/sessions";
...
```
- In this document, the namespace is 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**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>sessions:ids() as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the IDs of all registered sessions.</td>
</tr>
</tbody>
</table>

**sessions:created**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>sessions:created($id as xs:string) as xs:dateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the creation time of the session specified by $id.</td>
</tr>
</tbody>
</table>

**sessions:accessed**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>sessions:accessed($id as xs:string) as xs:dateTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the last access time of the session specified by $id.</td>
</tr>
</tbody>
</table>

**sessions:names**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>sessions:names($id as xs:string) as xs:string*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the names of all variables bound to the session specified by $id.</td>
</tr>
</tbody>
</table>

**sessions:get**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>sessions:get($id as xs:string, $name as xs:string) as item()*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sessions:get</td>
<td>sessions:get($id as xs:string, $name as xs:string, $default as item()<em>) as item()</em></td>
</tr>
</tbody>
</table>
Sessions Module

**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.

**Errors**

get: the value of an attribute could not be retrieved.

### 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.

**Errors**

set: The supplied value cannot be materialized.

### 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>get</td>
<td>The stored attribute is no XQuery value.</td>
</tr>
<tr>
<td>set</td>
<td>The supplied value cannot be materialized.</td>
</tr>
<tr>
<td>not-found</td>
<td>The specified session was not found.</td>
</tr>
</tbody>
</table>

### Changelog

**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 67. 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('dbc: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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>error</code></td>
<td>an error occurred while executing SQL</td>
</tr>
<tr>
<td><code>id</code></td>
<td>the specified connection does not exist</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>query execution exceeded timeout</td>
</tr>
</tbody>
</table>

#### `sql:execute-prepared`

<table>
<thead>
<tr>
<th>Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sql:execute-prepared($id as xs:anyURI, $params as element(sql:parameters))</code> as item()*</td>
</tr>
<tr>
<td><code>sql:execute-prepared($id as xs:anyURI, $params as element(sql:parameters), $options as map(*)?)</code> as item()*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function executes a prepared statement with the specified $id. The output format is identical to <code>sql:execute</code>. The optional parameter $params is an element <code>&lt;sql:parameters/&gt;</code> representing the parameters for a prepared statement along with their types and values. The following schema shall be used:</td>
</tr>
</tbody>
</table>

```xml
<element sql:parameters {
    <element sql:parameter {
        attribute type { "int" | "string" | "boolean" | "date" | "double" | "float" | "short" | "time" | "timestamp" | "sqlxml" },
        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

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attribute</code></td>
<td>an attribute different from type and null is set for a <code>&lt;sql:parameter/&gt;</code> element</td>
</tr>
<tr>
<td><code>error</code></td>
<td>an error occurred while executing SQL</td>
</tr>
<tr>
<td><code>id</code></td>
<td>the specified connection does not exist</td>
</tr>
<tr>
<td><code>parameters</code></td>
<td>no parameter type specified</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>query execution exceeded timeout</td>
</tr>
<tr>
<td><code>type</code></td>
<td>the value of a parameter cannot be converted to the specified format.</td>
</tr>
</tbody>
</table>

#### `sql:prepare`

<table>
<thead>
<tr>
<th>Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sql:prepare($id as xs:anyURI, $statement as xs:string)</code> as xs:anyURI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>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 <code>&lt;sql:parameter/&gt;</code> element must be true.</td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>error</code></td>
<td>an error occurred while executing SQL</td>
</tr>
<tr>
<td><code>id</code></td>
<td>the specified connection does not exist</td>
</tr>
</tbody>
</table>

#### `sql:commit`

<table>
<thead>
<tr>
<th>Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sql:commit($id as xs:anyURI)</code> as empty-sequence()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function commits the changes made to a relational database, using the specified connection $id.</td>
</tr>
</tbody>
</table>

### Errors

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>error</code></td>
<td>an error occurred while executing SQL</td>
</tr>
<tr>
<td><code>id</code></td>
<td>the specified connection does not exist</td>
</tr>
</tbody>
</table>

#### `sql:rollback`

<table>
<thead>
<tr>
<th>Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sql:rollback($id as xs:anyURI)</code> as empty-sequence()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>This function rolls back the changes made to a relational database, using the specified connection $id.</td>
</tr>
</tbody>
</table>
Errors  
error: an error occurred while executing SQL.

Error: 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">9.5</sql:column>
  <sql:column name="sales">15</sql:column>
  <sql:column name="total">30</sql:column>
</sql:row>
<sql:row xmlns:sql="http://basex.org/modules/sql">
  <sql:column name="cof_name">French_Roast_Decaf</sql:column>
  <sql:column name="sup_id">49</sql:column>
  <sql:column name="price">7.5</sql:column>
  <sql:column name="sales">10</sql:column>
  <sql:column name="total">14</sql:column>
</sql:row>
```

Prepared Statements

A prepared select statement can be executed in the following way:

```
(: Establish a connection :) 
let $conn := sql:connect("jdbc:postgresql://localhost:5432/coffeehouse")
(: Obtain a handle to a prepared statement :) 
let $prep := sql:prepare($conn, "SELECT * FROM coffees WHERE price < ? AND cof_name = ?")
(: Values and types of prepared statement parameters :) 
let $params := <sql:parameters>
  <sql:parameter type='double'>10</sql:parameter>
  <sql:parameter type='string'>French_Roast</sql:parameter>
</sql:parameters>
(: Execute prepared statement :) 
return sql:execute-prepared($prep, $params)
```

SQLite

The following expression demonstrates how SQLite can be addressed using the Xerial SQLite JDBC driver:
(; Initialize driver :)  sql:init("org.sqlite.JDBC"),
(; Establish a connection :)  let $conn := sql:connect("jdbc:sqlite:database.db")  return (  (: Create a new table :)  sql:execute($conn, "drop table if exists person"),  sql:execute($conn, "create table person (id integer, name string)")
  (: Run 10 updates :)  for $i in 1 to 10  let $q := "insert into person values($i, $i)"  return sql:execute($conn, $q),
  (: Return table contents :)  sql:execute($conn, "select * from person")
)

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 <code>&lt;sql:parameter/&gt;</code> 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>parameter</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 9.0

- 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 68. Strings Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions for string computations.

Conventions

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

Functions

### strings:levenshtein

**Signatures**

```
strings: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**

- `strings:levenshtein("flower", "flower")` returns 1
- `strings: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 $norm := ft:normalize(?,'stemming':true())
  return strings:levenshtein($norm("HOUSES"), $norm("house"))
  ```

### strings:soundex

**Signatures**

```
strings: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**

- `strings:soundex("Michael")` returns M240
- `strings:soundex("OBrien") = strings:soundex("O'Brien")` returns true

### strings:cologne-phonetic

**Signatures**

```
strings: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**

- `strings:cologne-phonetic("Michael")` returns 645
- `every $s in ("Mayr", "Maier", "Meier") satisfies strings:cologne-phonetic($s) = "67"` returns true
Changelog

The Module was introduced with Version 8.3.
Chapter 69. 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. 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.
Unit Module

Examples

• The first function will be evaluated before the actual test:

```xml
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>%unit:after %unit:after(FUNCTION)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>A function decorated with this annotation will be evaluated after each unit test. FUNCTION can be supplied as additional argument. It must be a valid EQName string. If specified, the function will only be evaluated after a function with the given name is tested.</td>
</tr>
</tbody>
</table>

%unit:before-module

<table>
<thead>
<tr>
<th>Syntax</th>
<th>%unit:before-module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>If a function is decorated with this annotation, it will be evaluated before all unit tests in the current module.</td>
</tr>
</tbody>
</table>

%unit:after-module

<table>
<thead>
<tr>
<th>Syntax</th>
<th>%unit:after-module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>If a function is decorated with this annotation, it will be evaluated after all unit tests in the current module.</td>
</tr>
</tbody>
</table>

%unit:ignore

<table>
<thead>
<tr>
<th>Syntax</th>
<th>%unit:ignore %unit:ignore(MESSAGE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>If a function is decorated with this annotation, it will temporarily be ignored by the test suite runner.</td>
</tr>
</tbody>
</table>

Functions

unit:assert

<table>
<thead>
<tr>
<th>Signatures</th>
<th>unit:assert($test as item(<em>) as empty-sequence()) unit:assert($test as item(</em>)*, $info as item()) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
<tr>
<td>Errors</td>
<td>fail: the assertion failed, or an error was raised.</td>
</tr>
</tbody>
</table>

unit:assert-equals

<table>
<thead>
<tr>
<th>Signatures</th>
<th>unit:assert-equals($returned as item()<em>, $expected as item()</em>) as empty-sequence() unit:assert-equals($returned as item()<em>, $expected as item()</em>, $info as item()) as empty-sequence()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 XQUnit module **tests.xqm** contains all available unit annotations:

**Query**

```xml
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/>
};
```

331
Unit Module

(:~ Function demonstrating the creation of a failure. :) 
declare %unit:test function test:failure() {
  unit:fail("Failure!"
);
}(~ Function demonstrating an error. :) 
declare %unit:test function test:error() {
  1 + <a/>
};
(:~ Skipping a test. :) 
declare %unit:test %unit:ignore("Skipped!") function test:skipped() {
  ()
};

By running TEST tests.xqm, the following report will be generated (timings may differ):

Result

<testsuites time="PT0.256S">
  <testsuite name="file:///C:/Users/user/Desktop/test.xqm" time="PT0.212S"
    tests="8" failures="4" errors="1" skipped="1">
    <testcase name="assert-success" time="PT0.016S"/>
    <testcase name="assert-failure" time="PT0.005S">
      <failure line="30" column="15">
        <info>Empty sequence.</info>
      </failure>
    </testcase>
    <testcase name="assert-equals-failure" time="PT0.006S">
      <failure line="35" column="22">
        <returned item="1" type="xs:integer">9</returned>
        <expected item="1" type="xs:integer">6</expected>
        <info>Item 1: 6 expected, 9 returned.</info>
      </failure>
    </testcase>
    <testcase name="unexpected-success" time="PT0.006S">
      <failure>
        <expected>FORG0001</expected>
      </failure>
    </testcase>
    <testcase name="expected-failure" time="PT0.004S"/>
    <testcase name="failure" time="PT0.004S">
      <failure line="50" column="13">
        <info>Failure!</info>
      </failure>
    </testcase>
    <testcase name="error" time="PT0.004S">
      <error line="55" column="6" type="FORG0001">
        <info>Cannot cast to xs:double: "".</info>
      </error>
    </testcase>
    <testcase name="skipped" skipped="Skipped!" time="PT0S"/>
  </testsuite>
</testsuites>

Errors

<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>
</tbody>
</table>
private | A test function must not be private.

**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

Version 7.8

• Added: unit:assert-equals
• Updated: enhanced test report output

This module was introduced with Version 7.7.
Chapter 70. 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.

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:

  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.

334
Update Module

Examples

• Renames nodes in an XML snippets:

```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

```xml
update:map-for-each($map as map(*), $function as function(xs:anyAtomicType, item()*) as 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

Updated with Version 9.1: Maps and arrays can be cached if they contain no persistent database nodes or function items.

Signatures

```xml
update:output($result as item()*) as empty-sequence()
```

Summary

This function is a helper function for returning results in an updating expression. The argument of the function will be evaluated, and the resulting items will be cached and returned after the updates on the pending update list have been processed. If the supplied item will be 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

```xml
update:cache() as item()*
```

Summary

Returns the items that have been cached by update:output. It 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 9.1

• 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 71. 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:

```xml
<user name="admin" permission="admin">
    <password algorithm="digest">
        <hash>304bdfb0383c16f070a897fc1eb25cb4</hash>
    </password>
    <password algorithm="salted-sha256">
        <salt>871602799292195</salt>
        <hash>a065ca66fa3d6da5762c227587f1c8258c6dc08ee867e44a605a72da115dcb41</hash>
    </password>
</user>
```

Errors unknown: The specified user name is unknown.

user:exists

Signatures user:exists($name as xs:string) as xs:boolean

Summary Checks if a user with the specified $name exists.
User Module

Examples
- user:exists('admin') will always yield true.

Errors
- name: The specified user name is invalid.

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
- user:check('admin', 'admin') will raise an error if the admin password was changed.

Errors
- name: The specified user name is invalid.
- unknown: The specified user does not exist.
- password: The specified password is wrong.

user:info

Signatures
- user:info() as element(info)

Summary
- Returns an info element, which can be used to organize application-specific data. By default, the element has no contents. It can be modified via user:update-info.

Examples
- After a fresh installation, user:info() returns <info/>.

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()

Summary
- Creates a new user with the specified $name, $password, and $permissions. Local permissions are granted with non-empty glob $patterns. The default global permission (none) can be overwritten with an empty pattern or by omitting the last argument. Existing users will be overwritten.

Examples
- user:create('John', '7e$j#!1', 'admin') creates a new user 'John' with admin permissions.
- user:create('Jack', 'top!secret', 'read', 'index*') creates a new user 'Jack' with no permissions, but write permissions for databases starting with the letters 'index'.

Errors
- name: The specified user name is invalid.
- permission: The specified permission is invalid.
- admin: The "admin" user cannot be modified.
- logged-in: The specified user is currently logged in.
- update: The operation can only be performed once per user or database pattern.

user:grant

Signatures
- user:grant($name as xs:string, $permissions 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 user name is unknown.
**name**: The specified user name is invalid.
**pattern**: The specified database pattern is invalid.
**permission**: The specified permission is invalid.
**admin**: The "admin" user cannot be modified.
**local**: A local permission can only be 'none', 'read' or 'write'.
**logged-in**: The specified user is currently logged in.
**update**: The operation can only be performed once per user or database pattern.

### user:drop

**Signatures**

```
user:drop($name as xs:string) as empty-sequence()
user:drop($name as xs:string, $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 dropped.

**Examples**

- `user:drop('John')` drops the user 'John'.
- `user:grant('John', 'unit*')` removes the 'unit*' database pattern. If John accesses any of these database, his global permission will be checked again.

**Errors**

**unknown**: The specified user name is unknown.
**name**: The specified user name is invalid.
**pattern**: The specified database pattern is invalid.
**admin**: The "admin" user cannot be modified.
**local**: A local permission can only be 'none', 'read' or 'write'.
**logged-in**: The specified user is currently logged in.
**update**: The operation can only be performed once per user or database pattern.

### user:alter

**Signatures**

```
user:alter($name as xs:string, $newname as xs:string) as empty-sequence()
```

**Summary**

Renames a user with the specified $name to $newname.

**Examples**

- `user:rename('John', 'Jack')` renames the user 'John' to 'Jack'.

**Errors**

**unknown**: The specified user name is unknown.
**name**: The specified user name is invalid.
**admin**: The "admin" user cannot be modified.
**logged-in**: The specified user is currently logged in.
**update**: The operation can only be performed once per user or database pattern.

### user:password

**Signatures**

```
user:password($name as xs:string, $password as xs:string) as empty-sequence()
```

**Summary**

Changes the password of a user with the specified $name.

**Examples**

- `user:password('John', )` assigns user 'John' an empty password string.

**Errors**

**unknown**: The specified user name is unknown.
**name**: The specified user name is invalid.
**update**: The operation can only be performed once per user or database pattern.

### user:update-info

**Signatures**

```
user:update-info($info as element(info)) as empty-sequence()
```

**Summary**

Updates the info element with $info. This node can be used to organize 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 }
})

• Assign a group to a new user:

let $user := 'john', $pw := '8hKJ8%./0O', $group := 'editor'
return (
    user:create($user, $pw),
    user:update-info(user:info() update
    insert node <user name='{ $user }' group='{ $group }'/> into .
    )
)

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 user name 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
• 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: user:create, user:grant, user:drop: extended support for database patterns.

Version 8.1
• Added: user:current.

The Module was introduced with Version 8.0.
Chapter 72. Validation Module

Read this entry online in the BaseX Wiki.

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 {</td>
</tr>
<tr>
<td></td>
<td>let $doc := &lt;invalid/&gt;</td>
</tr>
<tr>
<td></td>
<td>let $schema := '&lt;!ELEMENT root (#PCDATA)&gt;}'</td>
</tr>
<tr>
<td></td>
<td>return validate:dtd($doc, $schema)</td>
</tr>
<tr>
<td></td>
<td>} catch validate:error {</td>
</tr>
<tr>
<td></td>
<td>'DTD Validation failed.'</td>
</tr>
<tr>
<td></td>
<td>}</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>
</tbody>
</table>
Validation Module

Errors
- `init`: the validation process cannot be started.
- `not-found`: no DTD validator is available.

Examples
- `validate:dtd-info(<invalid/>, '<!ELEMENT root (#PCDATA)>')`
  returns: `2:11: Element type "invalid" must be declared.`

validate:dtd-report

width='100%'

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/>, '<!ELEMENT root (#PCDATA)>')`
  returns:

```
<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 implementations can be applied for validation:

- By default, the Java implementation of XML Schema 1.0 is used (it is based on an old version of Apache Xerces).
- If Xerces2 exists in the classpath, it will be used instead. If you want to manually add Xerces2, you should download the Xerces2 Java ..... (XML Schema 1.1) (Beta) archive and copy the four libraries (cupv10k-runtime.jar, org.eclipse.wst.xml.xpath2.processor_1.2.0.jar, xercesImpl.jar, xml-apis.jar) to the lib/custom directory of the full distribution of BaseX.
- The Saxon Enterprise Edition (saxon9ee.jar) is supported as well.

The XML Schema version can be specified as well (the supplied string must be 1.0 or 1.1). Please note that Version 1.1 is only available if Saxon EE or Xerces2 is added to the classpath.

validate:xsd

Signatures
- `validate:xsd($input as item()) as empty-sequence()`
- `validate:xsd($input as item(), $schema as item()?) as empty-sequence()`
- `validate:xsd($input as item(), $schema as item()?, $version as xs:string) as empty-sequence()`
Validation Module

**Summary**
Validates the XML $input document against a $schema, using the specified $version of XML Schema.

**Errors**
- **init**: the validation process cannot be started.
- **not-found**: no XML Schema validator is available.
- **version**: no validator is found for the specified version.

**Examples**
- `validate:xsd('doc.xml', 'doc.xsd')` validates the document `doc.xml` against the specified schema `doc.xsd`.
- The following example demonstrates how a document can be validated against a schema without resorting to local or remote URIs:

```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: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()?, $version as xs:string) as xs:string*`

**Summary**
Validates the XML $input document against a $schema, using the specified $version of XML Schema, 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.
- **version**: no validator is found for the specified version.

**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?, $version as xs:string) as element(report)`

**Summary**
Validates the XML $input document against a $schema, using the specified $version of XML Schema, 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.
- **version**: no validator is found for the specified version.

**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, simply install Vincent Lizzi’s excellent Schematron XQuery Module for BaseX:

```xml
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:

```xml
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($message), ': ', schematron:message-description($message)))
```

---

344
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>
<tr>
<td>version</td>
<td>No validator is found for the specified version.</td>
</tr>
</tbody>
</table>

Changelog

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 73. 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**

| Signatures | web:content-type($path as xs:string) as xs:string |
| Summary | Returns the content type of a path by analyzing its file suffix. application/octet-stream is returned if the file suffix is unknown. |
| Examples | • web:content-type("sample.mp3") returns audio/mpeg |

**web:create-url**

| Signatures | web:create-url($url as xs:string, $parameters as map(*)) as xs:string |
| Summary | Creates a new URL from the specified $url string and the $parameters specified in a map. The keys and values of the map entries will be converted to strings, URL-encoded (see web:encode-url), and appended to the url as query parameters. If a map entry has more than a single item, all of them will be appended as single parameters. |
| Examples | • web:create-url('http://find.me', map { 'q': 'dog' }) returns http://find.me?q=dog |

**web:encode-url**

| Signatures | web:encode-url($string as xs:string) as xs:string |
| Summary | Encodes a string to a URL. Spaces are rewritten to +; *,-, and _ are adopted; and all other non-ASCII characters and special characters are percent-encoded. |
| Examples | • web:encode-url("this is a test!.html") returns this+is+a+test%21.html. |

**web:decode-url**

| Signatures | web:decode-url($string as xs:string) as xs:string |
| Summary | Decodes a URL to the original string. Percent-encoded characters are decoded to their UTF8 codepoints, and + characters are rewritten to spaces. |
| Examples | • web:decode-url("%E6%97%A5%E6%9C%AC%E8%AA%9E") returns ###. |
| Errors | invalid: the string contains invalid XML characters. |
**web:redirect**

**Signatures**

`web:redirect($location as xs:string) as element(rest:response)`  
`web:redirect($location as xs:string, $parameters as map(*)) as element(rest:response)`

**Summary**

Creates a RESTXQ redirection to the specified location. The returned response will only work if no other items are returned by the RESTXQ function. If `$parameters` are specified, they will be appended as query parameters to the URL as described for `web:create-url`.

**Examples**

- The query `web:redirect('/a/b')` returns the following 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>
  ```

- The first RESTXQ function creates an initial database, and redirects to a second function that will access this database:

  ```xml
declare %updating %rest:path('/app/init') function local:init() {
  db:create('app', <root/>, 'root.xml'),
  db:output(web:redirect('/app/main'))
};
declare %rest:path('/app/main') function local:update() {
  'Stored documents: ' || count(db:open('app'))
};
```  

**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 top 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:
The following RESTXQ function returns the contents of a file to the client with correct media type:

```xquery
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)
  )
}
```

### 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>
</tbody>
</table>

### Changelog

**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 74. 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. The module must be imported in the query prolog:

  import module namespace ws = "http://basex.org/modules/ws";

- In this document, the namespace is 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 WebSocket.

**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.
- 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*  
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*  
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.

**WebSocket Attributes**

**ws:get**

*Signatures*  
ws:get($id as xs:string, $name as xs:string) 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.

*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.set: The supplied value cannot be materialized.

**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**
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

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.
• 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>set</td>
<td>The supplied value cannot be materialized.</td>
</tr>
<tr>
<td>not-found</td>
<td>No WebSocket with the specified id exists.</td>
</tr>
</tbody>
</table>

Changelog
This module was introduced with Version 9.1.
Chapter 75. 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.

Dynamic Evaluation

xquery:eval

Signatures

xquery:eval($query as xs:string) as item()*
xquery:eval($query as xs:string, $bindings as map(*)?) as item()*
xquery:eval($query as xs:string, $bindings as map(*)?, $options as map(*)?) as item()*

Summary

Evaluates the supplied $query string as XQuery expression and returns the resulting items. The evaluated query has its own query context. If a returned node is stored in a database, a main-memory copy will be returned as result, because the referenced database is closed after query execution and will not be accessible anymore. Variables and context items can be declared via $bindings. The specified keys must be QNames or strings:

- If a key is a QName, it will be directly adopted as variable name.
- It a key is a string, it may be prefixed with a dollar sign. Namespace can be specified using the Clark Notation.
- If the specified string is empty, the value will be bound to the context item.

The $options parameter contains evaluation options:

- 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. This URI 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.
- You can bind the context and e.g. operate on a certain database only:
xquery:eval("//country", map { '': db:open('factbook') })

- The following expressions use strings as keys. All of them return 'XML':

xquery:eval(".", map { '': 'XML' }),
xquery:eval("declare variable $xml external; $xml", map { 'xml': 'XML' }),
xquery:eval(
  "declare namespace pref='URI';
  declare variable $pref:xml external;
  $pref:xml",
  map { '{URI}xml': 'XML' }
)

- The following expressions use QNames as keys. All of them return 'XML':

declare namespace pref = 'URI';
xquery:eval("declare variable $xml external; $xml", map
  { xs:QName('xml'): 'XML' }),
let $query := "declare namespace pref='URI';
  declare variable $pref:xml external;
  $pref:xml"
let $vars := map { xs:QName('pref:xml'): 'XML' }
return xquery:eval($query, $vars)

xquery:eval-update

Signatures

xquery:eval-update($query as xs:string) as item()*
xquery:eval-update($query as xs:string, $bindings as map(*)?) as item()*
xquery:eval-update($query as xs:string, $bindings as map(*)?, $options as map(*)?) as item()

Summary

Evaluates $query as updating XQuery expression at runtime. All updates will be added to the Pending Update List of the main query and performed after the evaluation of the main query. The rules of the $bindings and $options parameters 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.

xquery:invoke

Signatures

xquery:invoke($uri as xs:string) as item()*
xquery:invoke($uri as xs:string, $bindings as map(*)?) as item()*
xquery:invoke($uri as xs:string, $bindings as map(*)?, $options as map(*)?) as item()*

Summary

Evaluates the XQuery module located at $uri at runtime and returns the resulting items. A relative URI will be resolved against the static base URI of the query. The rules of the $bindings and $options parameters are the same as for xquery:eval.

Errors

update: the expression contains updating expressions.
permission: insufficient permissions for evaluating the query.
timeout: query execution exceeded timeout.
nested: nested query evaluation is not allowed.
Any other error that may occur while evaluating the query.
### xquery:invoke-update

**Signatures**

|xquery:invoke-update($uri as xs:string) as item()*| xquery:invoke-update($uri as xs:string, $bindings as map(*))? as item()*
| xquery:invoke-update($uri as xs:string, $bindings as map(*)?, $options as map(*))? as item()*|

**Summary**

Evaluates the updating XQuery module located at $uri at runtime. A relative URI will be resolved against the static base URI of the query. The rules of the $bindings and $options parameters are the same as for xquery:eval.

**Errors**

update: the expression contains no updating expressions. permission: insufficient permissions for evaluating the query. timeout: query execution exceeded timeout. nested: nested query evaluation is not allowed. Any other error that may occur while evaluating the query.

### XQuery Parsing

### xquery:parse

**Signatures**

|xquery:parse($query as xs:string) as item()?| xquery:parse($query as xs:string, $options as map(*))? as item()?|

**Summary**

Parses the specified $query string as XQuery module and returns the resulting query plan. 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: passes on the original error info (line and column number, optional file uri). By default, this option is false.
- 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>
```

### xquery:parse-uri

**Signatures**

|xquery:parse-uri($uri as xs:string) as item()?| xquery:parse-uri($uri as xs:string, $options as map(*))? as item()?|

**Summary**

Parses the XQuery module located at $uri and returns the resulting query plan. A relative URI will be resolved against the static base URI of the query. The rules for the $options parameter are the same as for xquery:parse.

**Errors**

Any error that may occur while parsing the query.
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**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error</td>
<td>An unexpected error occurred.</td>
</tr>
<tr>
<td>permission</td>
<td>Insufficient permissions for evaluating the query.</td>
</tr>
<tr>
<td>update</td>
<td>Updating expression found or expected.</td>
</tr>
<tr>
<td>timeout</td>
<td>Query execution exceeded timeout.</td>
</tr>
<tr>
<td>memory</td>
<td>Query execution exceeded memory limit.</td>
</tr>
<tr>
<td>nested</td>
<td>Nested query evaluation is not allowed.</td>
</tr>
</tbody>
</table>

**Changelog**

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

• 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 76. XSLT Module

Read this entry online in the BaseX Wiki.

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.x of the Saxon XSLT Processor (saxon9he.jar, saxon9pe.jar, saxon9ee.jar) is found in the classpath (see Distributions for more details. A custom transformer can be specified by overwriting the system property javax.xml.transform.TransformerFactory, as shown in the following Java example:

```java
System.setProperty(
    "javax.xml.transform.TransformerFactory",
    "org.custom.xslt.TransformerFactoryImpl");

Context ctx = new Context();
String result = new XQuery("xslt:transform('...', '...')").execute(ctx);
...
ctx.close();
```

Conventions

All functions 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**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>xslt:processor() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the name of the applied XSLT processor, or the path to a custom implementation (currently: &quot;Java&quot;, &quot;Saxon EE&quot;, &quot;Saxon PE&quot;, or &quot;Saxon HE&quot;).</td>
</tr>
</tbody>
</table>

**xslt:version**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>xslt:version() as xs:string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Returns the supported XSLT version (currently: &quot;1.0&quot; or &quot;3.0&quot;). &quot;Unknown&quot; is returned if a custom implementation was chosen.</td>
</tr>
</tbody>
</table>

**xslt:transform**

<table>
<thead>
<tr>
<th>Signatures</th>
<th>xslt:transform($input as item(), $stylesheet as item()) as node()</th>
</tr>
</thead>
<tbody>
<tr>
<td>xslt:transform($input as item(), $stylesheet as item(), $params as map(*)?) as node()</td>
<td></td>
</tr>
<tr>
<td>xslt:transform($input as item(), $stylesheet as item(), $args as map(<em>)?, $options as map(</em>)?) as node()</td>
<td></td>
</tr>
</tbody>
</table>
| 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.

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 (currently, it is just a single one): |

357
XSLT Module

- cache: cache XSLT transformer (speeds up repeated transformations, but increases memory consumption)

Error

error: an error occurred during the transformation process.

**xslt:transform-text**

**Signatures**

```xml
xslt:transform-text($input as item(), $stylesheet as item()) as xs:string
xslt:transform-text($input as item(), $stylesheet as item(), $params as map(*)?) as xs:string
xslt:transform-text($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.

**Examples**

**Example 1: Basic XSL transformation with dummy document and without parameters**

**Query:**

```xml
<xslt:transform-text(<dummy/>, 'basic.xslt')
```

**Result:**

123

**Example 2: XSLT transformation of an input document**

**Query:**

```xml
let $in :=
<books>
  <book>
    <title>XSLT Programmer's Reference</title>
    <author>Michael H. Kay</author>
  </book>
  <book>
    <title>XSLT</title>
    <author>Doug Tidwell</author>
    <author>Simon St. Laurent</author>
    <author>Robert Romano</author>
  </book>
</books>
let $style :=
<xsl:stylesheet version='3.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
```

---

358
XSLT Module

```
<xsl:output method='xml'/>
<xsl:template match="/">
<html>
<body>
<div>
<xsl:for-each select='books/book'>
  • <b><xsl:apply-templates select='title'/></b>: <xsl:value-of select='author'/><br/>
</xsl:for-each>
</div>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

return xslt:transform($in, $style)

Result:

<html>
<body>
<div>
  • <b>XSLT Programmer’s Reference</b>: Michael H. Kay<br/>
  • <b>XSLT</b>: Doug Tidwell<br/>
</div>
</body>
</html>

Example 3: Assigning a variable to an XSLT stylesheet

Query:

let $in := <dummy/>
let $style := doc('variable.xsl')
return xslt:transform($in, $style, map { "v": 1 })

variable.xsl

```
xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xsl:param name='v'/>
<xsl:template match='/'>
  <v><xsl:value-of select='$v'/></v>
</xsl:template>
</xsl:stylesheet>

Result:

<v>1</v>
<v>1</v>

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 9.0
XSLT Module

- 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`
- Updated: `xslt:transform` returned error code

Version 7.3
- Updated: `$xslt:processor → xslt:processor, $xslt:version → xslt:version`
# Chapter 77. ZIP Module

Read this entry online in the BaseX Wiki.

This XQuery Module contains functions to handle ZIP archives. The contents of ZIP files can be extracted and listed, and new archives can be created. The module is based on the EXPath ZIP Module. Please note that the ZIP module is not being actively maintained but is still distributed for compatibility with older applications. We recommend you use the Archive Module wherever possible.

## Conventions

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

## Functions

**zip:binary-entry**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>zip:binary-entry($uri as xs:string, $path as xs:string) as xs:base64Binary</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Extracts the binary file at <code>$path</code> within the ZIP file located at <code>$uri</code> and returns it as an <code>xs:base64Binary</code> item.</td>
</tr>
<tr>
<td>Errors</td>
<td>ZIP0001: the specified path does not exist.ZIP0003: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**zip:text-entry**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>zip:text-entry($uri as xs:string, $path as xs:string) as xs:string</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>zip:text-entry($uri as xs:string, $path as xs:string, $encoding as xs:string) as xs:string</code></td>
</tr>
<tr>
<td>Summary</td>
<td>Extracts the text file at <code>$path</code> within the ZIP file located at <code>$uri</code> and returns it as an <code>xs:string</code> item. An optional encoding can be specified via <code>$encoding</code>.</td>
</tr>
<tr>
<td>Errors</td>
<td>ZIP0001: the specified path does not exist.ZIP0003: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**zip:xml-entry**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>zip:xml-entry($uri as xs:string, $path as xs:string) as document-node()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Extracts the XML file at <code>$path</code> within the ZIP file located at <code>$uri</code> and returns it as a document node.</td>
</tr>
<tr>
<td>Errors</td>
<td>ZIP0001: the specified path does not exist.ZIP0003: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**zip:html-entry**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>zip:html-entry($uri as xs:string, $path as xs:string) as document-node()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
<td>Extracts the HTML file at <code>$path</code> within the ZIP file located at <code>$uri</code> and returns it as a document node. The file is converted to XML first if Tagsoup is found in the classpath.</td>
</tr>
<tr>
<td>Errors</td>
<td>ZIP0001: the specified path does not exist.ZIP0003: the operation fails for some other reason.</td>
</tr>
</tbody>
</table>

**zip:entries**

<table>
<thead>
<tr>
<th>Signatures</th>
<th><code>zip:entries($uri as xs:string) as element(zip:file)</code></th>
</tr>
</thead>
</table>
ZIP Module

**Summary**
Generates a ZIP XML Representation of the hierarchical structure of the ZIP file located at $uri and returns it as an element node. The file contents are not returned by this function.

**Errors**
- ZIP0001: the specified path does not exist.
- ZIP0003: the operation fails for some other reason.

**Examples**
If the ZIP archive `archive.zip` is empty, `zip:entries('archive.zip')` returns:

```
```

**zip:zip-file**

**Signatures**
`zip:zip-file($zip as element(zip:file)) as empty-sequence()`

**Summary**
Creates a new ZIP archive with the characteristics described by `$zip`, the ZIP XML Representation.

**Errors**
- ZIP0001: an addressed file does not exist.
- ZIP0002: entries in the ZIP archive description are unknown, missing, or invalid.
- ZIP0003: the operation fails for some other reason.

**Examples**
The following function creates a file `archive.zip` with the file `file.txt` inside:

```
zip:zip-file(  
  <file xmlns="http://expath.org/ns/zip" href="archive.zip">  
    <entry src="file.txt"/>  
  </file>)
```

The following function creates a file `archive.zip`. It contains one file `readme` with the content "thanks":

```
zip:zip-file(  
  <file xmlns="http://expath.org/ns/zip" href="archive.zip">  
    <entry name="readme">thanks</entry>  
  </file>)
```

**zip:update-entries**

**Signatures**
`zip:update-entries($zip as element(zip:file), $output as xs:string) as empty-sequence()`

**Summary**
Updates an existing ZIP archive or creates a modified copy, based on the characteristics described by `$zip`, the ZIP XML Representation. The `$output` argument is the URI where the modified ZIP file is copied to.

**Errors**
- ZIP0001: an addressed file does not exist.
- ZIP0002: entries in the ZIP archive description are unknown, missing, or invalid.
- ZIP0003: the operation fails for some other reason.

**Examples**
The following function creates a copy `new.zip` of the existing `archive.zip` file:

```
zip:update-entries(zip:entries('archive.zip'), 'new.zip')
```

The following function deletes all PNG files from `archive.zip`:

```
declare namespace zip = "http://expath.org/ns/zip";
copy $doc := zip:entries('archive.zip')
modify delete node $doc//zip:entry[ends-with(lower-case(@name), '.png')]
return zip:update-entries($doc, 'archive.zip')
```

**Errors**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZIP0001</td>
<td>A specified path does not exist.</td>
</tr>
</tbody>
</table>
ZIP0002  Entries in the ZIP archive description are unknown, missing, or invalid.

ZIP0003  An operation fails for some other reason.
Part VII. Developing
Chapter 78. 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!

Project (Code, Features, Questions)

- The **Source Code** can be browsed online
- The **Issue Tracker** contains feature requests and bug reports
- For questions on the project, please write to our mailing list

Web Technology

- **RESTXQ**: Write web services with XQuery
- **REST**: Access and update databases via HTTP requests
- **WebDAV**: Access databases from your filesystem
- **XForms**: Build browser forms with XML technologies

APIs

- **Clients**: Communicate with BaseX using C#, PHP, Python, Perl, C, ...
- **Java Examples**: Code examples for developing with BaseX
- **XQJ API**, implemented by Charles Foster (closed source, restricted to XQuery 3.0)
- **XQuery for Scala API**, based on XQJ and written by Dino Fancellu

Extensions

- **Docker**: Isolate BaseX in a docker container
- **Service/daemon**: Install BaseX server as a service
- **Android**: Running BaseX with Android
Chapter 79. Developing with Eclipse

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to get the BaseX sources compiled and running on your system.

Another article in the documentation describes how to use BaseX as a query processor in Eclipse.

Prerequisites

BaseX is developed with the Eclipse environment (other IDEs like IntelliJ IDEA can be used as well). The Eclipse IDE for Java Developers includes the EGit plugin (for Git) and the m2e plugin (for Maven).

Other Eclipse plugins we use are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Update URL</th>
<th>Eclipse Marketplace</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindBugs</td>
<td>Analyze project at byte code level</td>
<td><a href="http://findbugs.cs.umd.edu/eclipse">http://findbugs.cs.umd.edu/eclipse</a></td>
<td>install</td>
</tr>
<tr>
<td>UCDetector</td>
<td>Unnecessary code detector</td>
<td><a href="http://ucdetector.sourceforge.net/updatedate">http://ucdetector.sourceforge.net/updatedate</a></td>
<td>install</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 sub-directories:

1. basex-core is the main project
2. basex-api contains the BaseX APIs (XML:DB, bindings in other languages) and HTTP Services (REST, RESTXQ, WebDAV)
3. basex-examples includes some examples code for BaseX
4. basex-tests contains several unit and stress tests

If the "Problems" View contains errors or warnings, you may need to switch to Java 7 (Windows → Preferences → Installed JREs). With the Maven plugin from Eclipse, it sometimes requires several attempts to get all dependencies updated. This loop can be avoided if the sources are precompiled via Maven on command-line.

Start in Eclipse

1. Press Run → Run...
2. Create a new "Java Application" launch configuration
3. Select "basex" as "Project"
4. Choose a "Main class" (e.g., org.basex.BaseXGUI for the graphical user interface)
5. Launch the project via *Run*

**Alternative**

You may as well use the standalone version of Maven to compile and run the project, use other IDEs such as IntelliJ IDEA.
Chapter 80. Git

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to use git to manage the BaseX sources.

Using Git to contribute to BaseX

Our team uses git and GitHub to manage the source code. All team members have read/write access to the repository, and external contributors are welcome to fork the project.

Git makes it easy to retain a full copy of the repository for yourself. To get started and running, simply fork BaseX:

1. Head over to https://github.com and create an account
2. Fork https://github.com/BaseXdb/basex, so you have a version on your own
3. The forked project can then be cloned on your local machine, and changes can be pushed back to your remote repository

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".

Create the project

- Select our newly cloned repository and click Next
- Select "Import Existing Projects" and depending on your Eclipse version enable automatic sharing. More recent versions will not offer this feature as sharing is enabled by default.
- Click next to select the Project to import
- Check "basex" to checkout and click finish
You are now ready to contribute.

EGit & SSH
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 Template:Cpde variable. According to this change in EGit, the plugin will try to use a native SSH implementation instead of JSch.

Using Git on Command-Line
Note: this is not intended to be a complete git reference; it's purpose is to quickly introduce BaseX developers to the most commonly used git commands in the context of the BaseX project.

Preparation
1. Create a GitHub user account: here (your github user name will be referenced as $username)
2. Set up SSH access to GitHub as described here
3. Create a fork of one of the BaseXdb projects (it will be referenced as $project)
4. Choose a directory where the project will be created and make it your working directory (e.g. /home/user/myprojects)

Clone Repository
$ git clone git@github.com:$username/$project.git
Cloning into $project...
Enter passphrase for key '/home/user/.ssh/id_rsa': ...

$ ls -d -l $PWD/*
/home/user/myprojects/$project

Note that git automatically creates a directory where the repository content will be checked out.

List Remote Repositories
$ git remote -v
origin git@github.com:$username/$project.git (fetch)
origin git@github.com:$username/$project.git (push)

Currently, there is only one remote repository; it is automatically registered during the clone operation. Git remembers this repository as the default repository for push/pull operations.
List Local Changes

After some files have been changed locally, the changes can be seen as follows:

```bash
$ 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:

```bash
$ git add readme.txt
```

Then perform the actual commit:

```bash
$ git commit
[master 0fde1fb] Added TODO in section "USING GIT"
 1 files changed, 4 insertions(+), 0 deletions(-)
```

Before executing the actual commit, git will open the default shell editor (determined using the $EDITOR variable, usually vi) to enter a message describing the commit changes.

Alternative way is to commit all changed files, i.e. it is not needed to explicitly add the changed files:

```bash
$ 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

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
• Comprehensive Getting Starting Guide on GitHub
• The git book
• Gitcasts.com – Video Guides
Chapter 81. Maven

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It demonstrates how Maven is used to compile and run BaseX, and embed it into other projects.

Using Maven

If you have cloned our repository and installed Maven on your machine, you can run the following commands from all local repository directories:

- `mvn compile`: the BaseX source files are compiled.
- `mvn package`: JAR archives are created in the target class directory, and all relevant libraries are created in the lib directory. Packaging is useful if you want to use the start scripts.
- `mvn install`: the JAR archive is installed to the local repository, and made available to other Maven projects. This is particularly useful if you are compiling a beta version of BaseX, for which no archives exist in the repositories.

By adding the flag `-DskipTests` you can skip the JUnit tests and speed up packaging. You may as well use Eclipse and m2eclipse to compile the BaseX sources.

There are several alternatives for starting BaseX:

- type in `java -cp target/classes org.basex.BaseX` in the basex-core directory to start BaseX on the command-line mode,
- type in `mvn jetty:run` in the basex-api directory to start BaseX with Jetty and the HTTP servers,
- run one of the Start Scripts contained in the etc directory

Artifacts

You can easily embed BaseX into your own Maven projects by adding the following XML snippets to your pom.xml file:

```
<repositories>
  <repository>
    <id>basex</id>
    <name>BaseX Maven Repository</name>
    <url>http://files.basex.org/maven</url>
  </repository>
</repositories>

<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex</artifactId>
  <version>7.6</version>
</dependency>

<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
</dependency>
```

APIs and Services

...including APIs and the REST, RESTXQ and WebDAV services:

```
<dependency>
  <groupId>org.basex</groupId>
  <artifactId>basex-api</artifactId>
</dependency>
```
Maven

<version>7.6</version>
</dependency>

**XQJ API**

The XQJ API is hosted at [http://xqj.net](http://xqj.net):

```xml
<repository>
  <id>xqj</id>
  <name>XQJ Maven Repository</name>
  <url>http://xqj.net/maven</url>
</repository>
...
<dependency>
  <groupId>net.xqj</groupId>
  <artifactId>basex-xqj</artifactId>
  <version>1.2.0</version>
</dependency>
<dependency>
  <groupId>com.xqj2</groupId>
  <artifactId>xqj2</artifactId>
  <version>0.1.0</version>
</dependency>
<dependency>
  <groupId>javax.xml.xquery</groupId>
  <artifactId>xqj-api</artifactId>
  <version>1.0</version>
</dependency>
```
Chapter 82. Releases

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It lists the official locations of major and minor BaseX versions:

**Official Releases**

Our releases, packaged for various platforms, are linked from our homepage. They are updated every 2-8 weeks:

- [http://basex.org/download](http://basex.org/download)

Our file server contains links to older releases as well (but we recommend everyone to stay up-to-date, as you’ll get faster feedback working with the latest version):

- [http://files.basex.org/releases](http://files.basex.org/releases)

**Stable Snapshots**

If you are a developer, we recommend you to regularly download one of our stable snapshots, which are packaged and uploaded several times a week:

- [http://files.basex.org/releases/latest/](http://files.basex.org/releases/latest/)

Note that the offered snapshot files are replaced as soon as newer versions are available.

**Code Base**

If you always want to be on the cutting edge, you are invited to watch and clone our GitHub repository:

- [https://github.com/BaseXdb/basex](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:

- [http://files.basex.org/maven/org/basex/](http://files.basex.org/maven/org/basex/)

**Linux**

BaseX can also be found in some Linux distributions, such as Debian, Ubuntu and archlinux (Suse and other distributions will follow soon):

- Debian: [http://packages.debian.org/sid/basex](http://packages.debian.org/sid/basex)
- Ubuntu: [http://launchpad.net/ubuntu/+source/basex](http://launchpad.net/ubuntu/+source/basex)
Chapter 83. 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 Dáníel
- **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 VIII. Web Technology
Chapter 84. RESTXQ

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the RESTXQ API of BaseX.

RESTXQ, introduced by Adam Retter, is an API that facilitates the use of XQuery as a server-side processing language for the Web. RESTXQ has been inspired by Java’s JAX-RS API: it defines a pre-defined set of XQuery 3.0 annotations for mapping HTTP requests to XQuery functions, which in turn generate and return HTTP responses.

Please note that BaseX provides various extensions to the 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

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 sub-directories will be traversed, and all XQuery files will be parsed for functions with RESTXQ annotations. Sub-directories 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:
module namespace page = 'http://basex.org/examples/web-page';

declare %rest:path("hello/{$who}") %rest:GET function page:hello($who) {
  <response>
    Hello {$who}!
  </response>
};

If the URI http://localhost:8984/hello/World is accessed, the result will be:

<response>
  <title>Hello World!</title>
</response>

The next function demonstrates a POST request:

declare
  %rest:path("/form")
  %rest:POST
  %rest:form-param("message","{$message}", "(no message)"
  %rest:header-param("User-Agent", "{$agent}"
function page:hello-postman(
  $message as xs:string,
  $agent as xs:string* ) as element(response) {
  <response type='form'>
    <message>{$message}</message>
    <user-agent>{$agent}</user-agent>
  </response>
};

If you post something (e.g. using curl or the embedded form at http://localhost:8984/)...

curl -i -X POST --data "message='CONTENT'" http://localhost:8984/form

...you will receive something similar to the following result:

HTTP/1.1 200 OK
Content-Type: application/xml; charset=UTF-8
Content-Length: 107
Server: Jetty(8.1.11.v20130520)

<response type="form">
  <message>'CONTENT'</message>
  <user-agent>curl/7.31.0</user-agent>
</response>

Request
This section shows how annotations are used to handle and process HTTP requests.

Constraints
Constraints restrict the HTTP requests that a resource function may process.

Paths
A resource function must have a single Path Annotation with a single string as argument. The function will be called if a URL matches the path segments and templates of the argument. Path templates contain variables in
curly brackets, and map the corresponding segments of the request path to the arguments of the resource function. The first slash in the path is optional.

The following example contains a path annotation with three segments and two templates. One of the function arguments is further specified with a data type, which means that the value for $variable will be cast to an xs:integer before being bound:

```xml
declare %rest:path("/a/path/{$with}/some/{$variable}")
    function page:test($with, $variable as xs:integer) { ... };
```

Variables can be enhanced by regular expressions:

```xml
(: Matches all paths with "app" as first, a number as second, and "order" as third segment :) 
declare %rest:path("app/{$code=[0-9]+}/order")
    function page:order($full-path) { ... };

(: Matches all other all paths starting with "app/" :) 
declare %rest:path("app/{$path=+.+}"
    function page:others($path) { ... };
```

### Content Negotiation

Two following annotations can be used to restrict functions to specific content types:

- **HTTP Content Types**: A function will only be invoked if the HTTP Content-Type header of the request matches one of the given content types. Example:

  ```xml
  %rest:consumes("application/xml", "text/xml")
  ```

- **HTTP Accept**: A function will only be invoked if the HTTP Accept header of the request matches one of the defined content types. Example:

  ```xml
  %rest:produces("application/atom+xml")
  ```

By default, both content types are */*. Quality factors supplied by a client will also be considered in the path selection process. If a client supplies the following accept header…

```xml
*/;q=0.5,text/html;q=1.0
```

…and if two RESTXQ functions exist with the same path annotation, one with the produces annotation */ *, and another with text/html, the second function will be called, because the quality factor for text/html documents is highest.

Server-side quality factors are supported as well: If multiple function candidates are left over after the above steps, the qs parameter will be considered. The function with the highest quality factor will be favored:

```xml
%rest:produces("text/html;qs=1")
%rest:produces("*/;qs=0.8")
```

Note that the annotation will not affect the content type of the actual response. You will need to supply an additional %output:media-type annotation.

### HTTP Methods

#### Default Methods

The HTTP method annotations are equivalent to all HTTP request methods except TRACE and CONNECT. Zero or more methods may be used on a function; if none is specified, the function will be invoked for each method.
The following function will be called if GET or POST is used as request method:

```xml
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:

```xml
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:

```xml
declare %rest:method("RETRIEVE")
  function page:retrieve() { "RETRIEVE was specified as request method." };
```

### Content Types

The body of a POST or PUT request will be converted to an XQuery item. Conversion can be controlled by specifying a content type. It can be further influenced by specifying additional content-type parameters:

<table>
<thead>
<tr>
<th>Content-Type</th>
<th>Parameters (<code>name=value</code>)</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>
<tr>
<td>text/html</td>
<td>HTML Options</td>
<td>document-node()</td>
</tr>
<tr>
<td>text/comma-separated-values</td>
<td>CSV Options</td>
<td>document-node() or map(*)</td>
</tr>
<tr>
<td>others</td>
<td></td>
<td>xs:base64Binary</td>
</tr>
<tr>
<td>multipart/*</td>
<td></td>
<td>sequence (see next paragraph)</td>
</tr>
</tbody>
</table>

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:
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="{$sum($add)}"/>
}
```

HTML Form Fields

Form parameters are specified the same way as query parameters. Their values are the result of HTML forms submitted with the content type application/x-www-form-urlencoded.

```xml
%rest:form-param("parameter", "{$value}", "default")
```

File Uploads

Files can be uploaded to the server by using the content type multipart/form-data (the HTML5 multiple attribute enables the upload of multiple files):

```xml
<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:

```xml
declare
%rest:POST
%rest:path("/upload")
%rest:form-param("files", "{$files}")
function page:upload($files) {
  for $name in map:keys($files)
    let $content := $files($name)
    let $path := file:temp-dir() || $name
    return (file:write-binary($path, $content),
      <file name="{$name}" size="{file:size($path)}"/>
    )
}
```
HTTP Headers

Header parameters are specified the same way as query parameters:

```xml
%rest:header-param("User-Agent", "{$user-agent}"
%rest:header-param("Referer", "{$referer}", "none"
```

Cookies

Cookie parameters are specified the same way as query parameters:

```xml
%rest:cookie-param("username", "{$user}
%rest:cookie-param("authentication", "{$auth}", "no_auth"
```

Query Execution

In many RESTXQ search scenarios, input from browser forms is processed and search results are returned. User experience can generally be made more interactive if an updated search request is triggered with each key click. However, this may lead to many expensive parallel 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 be executed for the same client. If the same function will be called for the second time, the already running query will be stopped, and the HTTP error code 460 will be returned instead:

```xml
(: If fast enough, returns the result. Otherwise, if called again, raises 460 :) declare    %rest:path("/search")
%rest:query-param("term", "{$term}"
%rest:single
function page:search($term as xs:string) {
   <ul>{
      for $result in db:open('large-db')/*[text() = $term]
      return <li>{$result}</li>
   }</ul>
};
```

By specifying a string along with the annotation, functions can be bundled together, and one request can be canceled by calling another one.

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>`.

```xml
declare    %rest:path("/compute")
%rest:single("EXPENSIVE")
function local:compute() {
   (1 to 100000000000000)[. = 0]
};

declare    %rest:path("/stop")
%rest:single("EXPENSIVE")
function local:stop() {
   <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 kill 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”).

**Custom Response**

Custom responses can be generated in XQuery by returning a `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:

```xquery
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**

The two XML elements `rest:forward` and `rest:redirect` can be used in the context of RESTXQ. These nodes allow e.g. multiple XQuery updates in a row by redirecting to the RESTXQ path of updating functions. Both wrap a URL to a RESTXQ path. The wrapped URL should be properly encoded via `fn:encode-for-uri()`.

Note that, currently, these elements are not part of RESTXQ specification.

**rest:forward**

Usage: wrap the location as follows

```xquery
<rest:forward>{ $location }</rest:forward>
```

This results in a server-side forwarding, which as well reduces traffic among client and server. A forwarding of this kind will not change the URL seen from the client's perspective.

As an example, returning

```xquery
<rest:forward>/hello/universe</rest:forward>
```

would internally forward to `http://localhost:8984/hello/universe`

**rest:redirect**

The function `web:redirect` can be used to create a redirect response element. Alternatively, the following element can be sent:
<rest:redirect>{ $location }</rest:redirect>

It is an abbreviation for:

<rest:response>
  <http:response status="302">
    <http:header name="location" value="{ $location }">
  </http:response>
</rest:response>

The client decides whether to follow this redirection. Browsers usually will, tools like curl won’t unless `-L` is specified.

Output

The content-type of a response can be influenced by the user via Serialization Parameters. The steps are described in the REST chapter. In RESTXQ, serialization parameters can be specified in the query prolog, via annotations, or within the REST response element:

Query Prolog

In main modules, serialization parameters may be specified in the query prolog. These parameters will then apply to all functions in a module. In the following example, the content type of the response is overwritten with the `media-type` parameter:

```xml
declare option output:media-type 'text/plain';

declare %rest:path("version1") function page:version1() {
  'Keep it simple, stupid'
};
```

Annotations

Global serialization parameters can be overwritten via `%output` annotations. The following example serializes XML nodes as JSON, using the JsonML format:

```xml
declare
  %rest:path("cities")
  %output:method("json")
  %output:json("format=jsonml")
function page:cities() {
  element cities {
    db:open('factbook')//city/name
  }
};
```

The next function, when called, generates XHTML headers, and `text/html` will be set as content type:

```xml
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 response element in a query. Serialization parameters will be overwritten:

```xml
declare %rest:path("version3") function page:version3() {
  <rest:response>
    <output:serialization-parameters>
      <output:media-type value='text/plain'/>
    </output:serialization-parameters>
  </rest:response>,
  'Not that simple anymore'
}
```

Error Handling

XQuery Errors

XQuery runtime errors can be processed via error annotations. Error annotations have one or more arguments, which represent the error codes to be caught. The codes equal the names of the XQuery 3.0 try/catch construct:

<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
declare %rest:path("/check//{$user}")
function page:check($user) {
  if($user = ('jack', 'lisa'))
    then 'User exists'
  else fn:error(xs:QName('err:user'), $user)
}
```

```xml
declare %rest:error("err:user")
%rest:error-param("description", "{$user}")
function page:user-error($user) {
```

387
An XQuery error in a RESTXQ context delivers by default a HTTP status code 400 error back to the client. However, you can also define a custom error code by using the third argument of the error function:

```
declare
  %rest:path("/teapot")
function page:teapot() {
  fn:error(xs:QName('error'), "I'm a teapot", 418)
};
```

### HTTP Errors

Errors that occur outside RESTXQ can be caught by adding `error-page` elements with an error code and a target location to the `web.xml` configuration file (find more details in the Jetty Documentation):

```
<error-page>
  <error-code>404</error-code>
  <location>/error404</location>
</error-page>
```

The target location may be another RESTXQ function. The `request:attribute` function can be used to request details on the caught error:

```
declare %rest:path("/error404") function page:error404() {
  "URL: " || request:attribute("javax.servlet.error.request_uri") || ", " ||
  "Error message: " || request:attribute("javax.servlet.error.message")
};
```

### User Authentication

If you want to provide restricted access to parts of a web application, 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 URIs and generating a WADL description of all services. Please note that the namespaces of all of these modules must be explicitly specified via module imports in the query prolog.

The following example returns the current host name:

```
import module namespace request = "http://exquery.org/ns/request";

declare %rest:path("/host-name") function page:host() {
  'Remote host name: ' || request:remote-hostname()
};
```

### References

Documentation:

- RESTXQ Specification, First Draft
- RESTful XQuery, Standardised XQuery 3.0 Annotations for REST, Paper, XMLPrague, 2012
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 9.0

- 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 85. Permissions

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:

```xml
(% 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() {
```
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,...).

  *Added with Version 9.1:* authorization: Value of the HTTP Authorization header string; may be empty.

An example:

```xml
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('/')
}
```

391
Permissions

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 (user name, 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:

```
declare
%rest:path("/login-check")
%rest:query-param("name", 
%rest:query-param("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 86. 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 [polling https://en.wikipedia.org/wiki/Polling_(computer_science)]. 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 sub-directories will be traversed, and all XQuery files will be parsed for functions with WebSocket annotations. Sub-directories 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
declare
  %ws:header-param('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>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 URI.</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.
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("/"):

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:

var ws = new WebSocket("ws://localhost:8984/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 87. REST

Read this entry online in the BaseX Wiki.

This page presents one of the Web Application services. It describes how to use the REST API of BaseX.

BaseX offers a RESTful API for accessing database resources via URLs. REST (REpresentational State Transfer) facilitates a simple and fast access to databases through HTTP. The HTTP methods GET, PUT, DELETE, and POST can be used to interact with the database.

Usage

By default, REST services are available at http://localhost:8984/rest/. If no default credentials are specified in the URL or when starting the web application, they will be requested by the client (see further).

A web browser can be used to perform simple GET-based REST requests and display the response. Some alternatives for using REST are listed in the Usage Examples.

URL Architecture

The root URL lists all available databases. The following examples assume that you have created a database instance from the factbook.xml document:

   http://localhost:8984/rest

   <rest:databases resources="1" xmlns:rest="http://basex.org/rest">
      <rest:database resources="1" size="1813599">factbook</rest:database>
   </rest:databases>

The resources of a database can be listed by specifying the database, and potential sub directories, in the URL. In the given example, a single XML document is stored in the factbook database:

   http://localhost:8984/rest/factbook

   <rest:database name="factbook" resources="1" xmlns:rest="http://basex.org/rest">
      <rest:resource type="xml" content-type="application/xml" size="77192">factbook.xml</rest:resource>
   </rest:database>

The contents of a database can be retrieved by directly addressing the resource:

   http://localhost:8984/rest/factbook/factbook.xml

If a resource is not found, an HTTP response will be generated with 404 as status code.

Parameters

The following parameters can be applied to the operations:

- **Variables**: External variables can be *bound* before a query is evaluated (see below for more).
- **Context**: The context parameter may be used to provide an initial XML context node.
- **Options**: Specified Options are applied before the actual operation will be performed.
- **Serialization**: All Serialization parameters known to BaseX can be specified as query parameters. Parameters that are specified within a query will be interpreted by the REST server before the output is generated.
While **Options** can be specified for all operations, the remaining parameters will only make sense for **Query** and **Run**.

**Request**

**GET Method**

If the GET method is used, all query parameters are directly specified within the URL. Additionally, the following operations can be specified:

- **query**: Evaluate an XQuery expression. If a database or database path is specified in the URL, it is used as initial query context.
- **command**: Execute a single database command.
- **run**: Evaluate an XQuery file or command script located on the server. The file path is resolved against the directory specified by **RESTPATH** (before, it was resolved against **WEBPATH**).

**Examples**

- Lists all resources found in the **tmp** path of the **factbook** database:
  ```
  http://localhost:8984/rest/factbook/tmp
  ```

- Serializes a document as JSONML:
  ```
  ```

- 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'/>
    <info-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>
</tbody>
</table>
REST

<table>
<thead>
<tr>
<th>parameter</th>
<th>Serialization parameter (with @name and @value attributes)</th>
</tr>
</thead>
<tbody>
<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:query xmlns:rest="http://basex.org/rest">
    <rest:text><![CDATA[(//city/name)[position() <= 5]]]></rest:text>
  </rest:query>
  ```

- 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>
  </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.

- **Store Resource**: A resource is added to the database if the URL contains a database path. If the addressed resource already exists, it is replaced by the new input.
There are two ways to store non-XML data in BaseX:

- **Store as Raw 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
POST Method

If query or run is used as operation, external variables can be specified via the <variable/> element:

```xml
<query xmlns="http://basex.org/rest">
  <text><![CDATA[
    declare variable $x as xs:integer external;
    declare variable $y as xs:integer external;
    <mult>{ $x * $y }</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:

```
http://localhost:8984/rest?query=%3Ca/%3E
http://localhost:8984/rest?query=%3Ca/%3E&method=xml
http://localhost:8984/rest?query=%3Ca/%3E&mediatype=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 user name and password pair with a base64 implementation.
        String encoded = Base64.encode(user + ":" + pw);
        // Basic access authentication header to connection request.
        conn.setRequestProperty("Authorization", "Basic " + encoded);

        // Print the HTTP response code.
        System.out.println("HTTP response: "+ conn.getResponseCode());

        // Close connection.
        conn.disconnect();
    }
}
```

Content-Types

The content-type of the input can easily be included, just add the following property to the connection (in this example we explicitly store the input file as raw):

```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"

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 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 addresses resources with application/xquery as content type

Version 7.1.1

• Added: options parameter for specifying database options

Version 7.1

• Added: PUT request: automatic conversion to XML if known content type is specified

Version 7.0

• REST API introduced, replacing the old JAX-RX API
Chapter 88. WebDAV

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).

Authorization

The WebDAV service uses the database user credentials in order to perform authentication and authorization. If database user and password are explicitly specified when starting the BaseX HTTP Server using the corresponding startup options, WebDAV will not request additional user authentication from the client.

Root Directory

In the WebDAV root directory, all existing databases are listed. As new resources can only be stored 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 http://www.w3schools.com/charsets/ref_utf_punctuation.asp)

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.
WebDAV

**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 89. WebDAV: Windows 7

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows 7.

- Open the Explorer
- Open the "Map network drive..." dialog by right-clicking on "My Computer"
- Click on the link "Connect to a Web site that you can use to store your documents and pictures."

- Click "Next", select "Choose a custom network location" and click "Next" again.

- Enter the URL address of the BaseX WebDAV Server (e.g. http://localhost:8984/webdav) and click "Next".
WebDAV: Windows 7

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 90. WebDAV: Windows XP

Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with Windows XP.

- In the "My Network Places" view, double click on "Add Network Place":

![Image of Add Network Place dialog]

- Confirm the upcoming introductory dialog:

![Image of Welcome to the Add Network Place Wizard]

- Select "Choose another network location" in the next dialog:
• Next, specify the BaseX WebDAV URL:

• Enter the user/password combination to connect to the WebDAV service:
• Assign a name to your WebDAV connection:

• Finish the wizard:

• You can now see all BaseX databases in the Windows Explorer:
Chapter 91. 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:
Chapter 92. WebDAV: GNOME

Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with GNOME and Nautilus.

• In Nautilus choose File -> Connect to Server:

• Choose "WebDAV (HTTP)" from the "Type" drop-down and enter the server address, port and user credentials:
After clicking "Connect" the databases can be browsed:

![WebDAV: GNOME](image_url)

- Connect to Server
- Server Details
  - Server: localhost
  - Port: 8984
  - Type: WebDAV (HTTP)
  - Folder: /webdav
- User Details
  - User name: admin
  - Password: ********
  - Remember this password
- Connect button

- After clicking "Connect" the databases can be browsed:

![Database Browser](image_url)

- data directory
- webdav
- myproject
- data
  - records.xml

1 item
Chapter 93. WebDAV: KDE

Read this entry online in the BaseX Wiki.

This page belongs to the WebDAV page. It describes how to get the WebDAV API running with KDE.

• KDE SC provides two file managers - Dolphin and Konqueror, which both support WebDAV using the "webdav://" URL prefix. Start Dolphin or Konqueror and enter the BaseX WebDAV URL (eg. webdav://localhost:8984/webdav):

• Enter the user credentials:

• After clicking "OK" the databases can be browsed:
Chapter 94. XForms

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section and belongs to the Web Application stack. It describes how to use XForms with BaseX.

XForms provides mechanisms to display and edit the contents of XML snippets in the browser without resorting to JavaScript. In combination with the RESTXQ API and the database backend, this is an elegant way of building web applications that completely reside in the XML stack.

Internals

If an HTML document with XForms elements is requested, a web form is generated, which allows users to edit the contents of an XML file. The data model stays consistent during this process, i.e. the data types are always as described in the models. Actions can be configured as well: the model can e. g. be sent to a server and processed further, using the REST or RESTXQ APIs.

Run the example

Several implementations of the XForms Recommendation are available (some AJAX-based, some client-side). In this article, we will focus on the light-weight, LGPL-licensed XSLTForms project from Alain Couthures. The following steps are required to get the XForms example running:

- download the xsltforms.zip example file, which includes the demo page and XSLTForms, and extract its contents to your webapp directory
- start a BaseX HTTP server
- open a browser and visit the URL http://localhost:8984/static/xforms-demo.xml

This is the head section of the XForms demo:

```xml
<?xml-stylesheet href="xsltforms/xsltforms.xsl" type="text/xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xf="http://www.w3.org/2002/xforms">
  <!-- ... -->
</html>
```

The processing instruction in the first line tells the browser where to find the XSLTForms implementation. The rules of the XML Stylesheet will then be applied to transform the XForms elements in the document to HTML. As the XForms code is interpreted, it can not be inspected in the browser, but you can press F1 to enter the debug mode.

Usually, the XForms Model is placed in the head section of the HTML document:

```xml
<xf:model>
  <xf:instance>
    <track xmlns="">
      <metadata>
        <title>Like A Rolling Stone</title>
        <artist>Bob Dylan</artist>
        <date>1965-06-21</date>
        <genre>Folk</genre>
      </metadata>
      <xf:bind id="_date" nodeset="//date" type="xs:date"/>
    </track>
  </xf:instance>
</xf:model>
```
It contains an instance of the model and a binding. The model is some plain XML, and the xf:bind elements can be used to bind elements to a specific type.

The data can be accessed with the xf:output element, and the XML nodes to be displayed are addressed via XPath 1.0 in the ref attribute. For example, the artist is addressed via:

```
<xf:output ref="//artist"/>
```

To modify the XML instance, xf:input elements are used. With the following code,

```
<xf:input ref="//date" incremental="true"/>
```

an input element is displayed that allows users to change the date. As xs:date was bound to dates in the data model, a date picker will be presented for choosing a valid date.

**Further reading**

For further reading, you are invited to

- check out the [XForms Wikibooks](#) page,
- look into the [XForms 2.0](#) specification, or
- join the [xsltforms-support](#) mailing list.
Part IX. Client APIs
Chapter 95. Clients

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. It describes how to communicate with BaseX from other programming languages.

You can use the following light-weight language bindings to connect to a running BaseX server instance, execute database commands and evaluate XQuery expressions.

Most clients provide two modes:

- **Standard Mode**: connecting to a server, sending commands
- **Query Mode**: defining queries, binding variables, iterative evaluation

Please see the Server Protocol for more information on the available commands. Currently, we offer bindings for the following programming languages:

<table>
<thead>
<tr>
<th>BaseX 7.x, BaseX 8.x and later</th>
<th>BaseX 7.x (outdated)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java : The default implementation</td>
<td>ActionScript : contributed by Manfred Knobloch</td>
</tr>
<tr>
<td>C++ : contributed by Jean-Marc Mercier</td>
<td>Haskell : contributed by Leo Wörteler</td>
</tr>
<tr>
<td>C# , contributed by the BaseX Team and Martin Ferrari</td>
<td>Lisp : contributed by Andy Chambers</td>
</tr>
<tr>
<td>C , contributed by the BaseX Team</td>
<td>node.js : contributed by Hans Hübner (deviating from client API)</td>
</tr>
<tr>
<td>Golang : contributed by Christian Baune</td>
<td>Qt : contributed by Hendrik Strobelt</td>
</tr>
<tr>
<td>Erlang : contributed by Zachary Dean</td>
<td>Rebol : contributed by Sabu Francis</td>
</tr>
<tr>
<td>node.js : contributed by Andy Bunce</td>
<td>Scala : contributed by Manuel Bernhardt</td>
</tr>
<tr>
<td>Perl , contributed by the BaseX Team</td>
<td>Scala (simple implementation)</td>
</tr>
<tr>
<td>PHP : updated by James Ball</td>
<td>VB , contributed by the BaseX Team</td>
</tr>
<tr>
<td>Python : contributed by Hiroaki Itoh</td>
<td></td>
</tr>
<tr>
<td>Python , using BaseX REST services: contributed by Luca Lianas</td>
<td></td>
</tr>
<tr>
<td>R : contributed by Ben Engbers</td>
<td></td>
</tr>
<tr>
<td>Ruby , contributed by the BaseX Team</td>
<td></td>
</tr>
</tbody>
</table>

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 8.0

• Updated: cram-md5 replaced with digest authentication
Chapter 96. 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", "admin");

    // perform command and print returned string
    print $session->execute("xquery 1 to 10");

    // close session
    $session->close();

    // print time needed
    $time = (microtime(true) - $start) * 1000;
    print "$time ms\n";
} catch (Exception $e) {
    // print exception
    print $e->getMessage();
}
?>
```
Chapter 97. Query Mode

Read this entry online in the BaseX Wiki.

The query mode of the Clients allows you to bind external variables to a query and evaluate the query in an iterative manner. The query() function of the Session instance returns a new query instance.

Usage

The query execution works as follows:

1. Create a new session instance with hostname, port, username and password.
2. Call query() with your XQuery expression to get a query object.
3. Optionally bind variables to the query with one of the bind() functions.
4. Optionally bind a value to the context item via context().
5. Iterate through the query object with the more() and next() functions.
6. As an alternative, call execute() to get the whole result at a time.
7. info() gives you information on query evaluation.
8. options() returns the query serialization parameters.
9. Don’t forget to close the query with close().

PHP Example

Taken from our repository:

```php
<?php
/*
 * This example shows how queries can be executed in an iterative manner.
 * Documentation: http://basex.org/api
 * (C) BaseX Team 2005-15, BSD License
*/
include("BaseXClient.php");

try {
    // create session
    $session = new Session("localhost", 1984, "admin", "admin");

    try {
        // create query instance
        $input = 'declare variable $name external; ' .
            'for $i in 1 to 10 return element { $name } { $i }';
        $query = $session->query($input);

        // bind variable
        $query->bind("$name", "number");

        // print result
        print $query->execute()."\n";

        // close query instance
        $query->close();
    }
}
```
Changelog

Version 7.2

- Added: context() function
Chapter 98. Server Protocol

This page presents the classes and functions of the BaseX Clients, and the underlying protocol, which is utilized for communicating with the database server. A detailed example demonstrates how a concrete byte exchange can look like.

Workflow

- All clients are based on the client/server architecture. Hence, a BaseX database server must be started first.
- Each client provides a session class or script with methods to connect to and communicate with the database server. A socket connection will be established by the constructor, which expects a host, port, user name and password as arguments.
- The `execute()` method is called to launch a database command. It returns the result or throws an exception with the received error message.
- The `query()` method creates a query instance. Variables and the context item can be bound to that instance, and the result can either be requested via `execute()`, or in an iterative manner with the `more()` and `next()` functions. If an error occurs, an exception will be thrown.
- The `create()`, `add()`, `replace()` and `store()` method pass on input streams to the corresponding database commands.
- To speed up execution, an output stream can be specified by some clients; this way, all results will be directed to that output stream.
- Most clients are accompanied by some example files, which demonstrate how database commands can be executed or how queries can be evaluated.

Transfer Protocol

All Clients use the following client/server protocol to communicate with the server. The description of the protocol is helpful if you want to implement your own client.

Conventions

- \x: single byte.
- {...}: utf8 strings or raw data, suffixed with a \00 byte. To avoid confusion with this end-of-string byte, all transferred \00 and \FF bytes are prefixed by an additional \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: {realm:nonce}
3. Client sends the user name and a hash value. The hash is composed of the md5 hash of
   a. the md5 hash of the user name, realm, and password (all separated by a colon), and
   b. the nonce: {username} {md5(md5(username:realm:password) + nonce)}
4. Server replies with \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 user name and a hash value. The hash is composed of the md5 hash of
   a. the md5 of the password and
   b. the nonce: \{username\} \{md5(md5(password) + nonce)\}
4. Server replies with \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}</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}</td>
<td>{input} \00</td>
<td>Creates a new database with the specified input (may be empty).</td>
</tr>
<tr>
<td>ADD</td>
<td>\09 {name}</td>
<td>{path} \00</td>
<td>Adds a new resource to the opened database.</td>
</tr>
<tr>
<td>REPLACE</td>
<td>\0C {path}</td>
<td>{input} \00</td>
<td>Replaces a resource with the specified input.</td>
</tr>
<tr>
<td>STORE</td>
<td>\0D {path}</td>
<td>{input} \00</td>
<td>Stores 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>
<tr>
<td>BIND</td>
<td>\03 {id}</td>
<td>\00 \00</td>
<td>Binds a value to a variable. The type will be ignored if the string is empty.</td>
</tr>
</tbody>
</table>
| RESULTS       | \04 \{id\}   | \xx \{item\} ... \xx \{item\} \00 | Returns all resulting items as strings, prefixed by a single byte (\xx) that
represents the Type ID. This command is called by the \more{} function of a client implementation.

\begin{tabular}{|l|l|}
\hline
\textbf{EXECUTE} & \texttt{\05 \{id\} \{result\} \00} \quad \text{Executes the query and returns the result as a single string.} \\
\hline
\textbf{INFO} & \texttt{\06 \{id\} \{result\} \00} \quad \text{Returns a string with query compilation and profiling info.} \\
\hline
\textbf{OPTIONS} & \texttt{\07 \{id\} \{result\} \00} \quad \text{Returns a string with all query serialization parameters, which can e.g. be assigned to the \textsc{serializer} option.} \\
\hline
\textbf{CONTEXT} & \texttt{\0E \{id\} \{value\} \00 \00 \{type\}} \quad \text{Binds a value to the context. The type will be ignored if the string is empty.} \\
\hline
\textbf{UPDATING} & \texttt{\1E \{id\} \{result\} \00} \quad \text{Returns true if the query contains updating expressions; false otherwise.} \\
\hline
\textbf{FULL} & \texttt{\1F \{id\} XDM \{item\} ... XDM \{item\} \00} \quad \text{Returns all resulting items as strings, prefixed by the XDM Meta Data. This command is e.g. used by the XQJ API.} \\
\hline
\end{tabular}

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 \texttt{error} message string.

\textbf{Binding Sequences}

Also sequences can be bound to variables and the context:

- \texttt{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 (\texttt{xs:integer} may be specified via the \{type\} argument).

- the two items \texttt{xs:integer(123)} and \texttt{xs:string('ABC')} are encoded as 123, \02, \texttt{xs:integer}, \01, \texttt{ABC}, \02, \texttt{xs:string} and \00.

\textbf{Example}

In the following example, a client registers a new session and executes the \texttt{INFO} database command. Next, it creates a new query instance for the XQuery expression \texttt{1, 2+3}. The query is then evaluated, and the server returns the result of the first subexpression \texttt{1} and an error for the second sub expression. Finally, the query instance and client session are closed.
Client connects to the database server socket

Server sends realm and timestamp "BaseX:1369578179679": # 42 61 73 65 58 3A 31 33 36 39
35 37 38 31 37 39 36 37 39 00

Client sends user name "jack": 6A 61 63 6B 00 #

Client sends hash: md5(md5("jack:BaseX:topsecret") + "1369578179679") = "ca664a31f8deda9b71ea3e79347f6666": 63 61 36 ... 00 #

Server replies with success code: # 00

Client sends the "INFO" command: 49 4E 46 4F 00 #

Server responds with the result "General Information...": # 47 65 6e 65 ... 00

Server additionally sends an (empty) info string: # 00

Client creates a new query instance for the XQuery "1, 2+3": 00 31 2C 20 32 2B 27 33 27 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 sends 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 ... 00

Client closes the query instance: 02 31 00 #

Server sends a response (which is equal to an empty info string) and success code: # 00 00

Client closes the socket connection

Constructors and Functions

Most language bindings provide the following constructors and functions:

**Session**

Create and return session with host, port, user name and password: 
```
Session(String host, int port, String name, String password)
```

Execute a command and return the result:
```
String execute(String command)
```

Return a query instance for the specified query:
```
Query query(String query)
```

Create a database from an input stream:
```
void create(String name, InputStream input)
```

Add a document to the current database from an input stream:
```
void add(String path, InputStream input)
```

Replace a document with the specified input stream:
```
void replace(String path, InputStream input)
```

Store raw data at the specified path:
```
void store(String path, InputStream input)
```

Return process information:
```
String info()
```

Close the session:
```
void close()
```

431
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 99. Server Protocol: Types

Read this entry online in the BaseX Wiki.

This article lists extended type information that is returned by the Server Protocol.

XDM Meta Data

In most cases, the XDM meta data is nothing else than the Type ID. There are three exceptions: 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>function</td>
</tr>
<tr>
<td>8</td>
<td>node()</td>
<td>node</td>
</tr>
<tr>
<td>9</td>
<td>text()</td>
<td>node</td>
</tr>
<tr>
<td>10</td>
<td>processing-instruction()</td>
<td>node</td>
</tr>
<tr>
<td>11</td>
<td>element()</td>
<td>node</td>
</tr>
<tr>
<td>12</td>
<td>document-node()</td>
<td>node</td>
</tr>
<tr>
<td>13</td>
<td>document-node(element())</td>
<td>node</td>
</tr>
<tr>
<td>14</td>
<td>attribute()</td>
<td>node</td>
</tr>
<tr>
<td>15</td>
<td>comment()</td>
<td>node</td>
</tr>
<tr>
<td>32</td>
<td>item()</td>
<td>atomic value</td>
</tr>
<tr>
<td>33</td>
<td>xs:untyped</td>
<td>atomic value</td>
</tr>
<tr>
<td>34</td>
<td>xs:anyType</td>
<td>atomic value</td>
</tr>
<tr>
<td>35</td>
<td>xs:anySimpleType</td>
<td>atomic value</td>
</tr>
<tr>
<td>36</td>
<td>xs:anyAtomicType</td>
<td>atomic value</td>
</tr>
<tr>
<td>37</td>
<td>xs:untypedAtomic</td>
<td>atomic value</td>
</tr>
<tr>
<td>38</td>
<td>xs:string</td>
<td>atomic value</td>
</tr>
<tr>
<td>39</td>
<td>xs:normalizedString</td>
<td>atomic value</td>
</tr>
<tr>
<td>40</td>
<td>xs:token</td>
<td>atomic value</td>
</tr>
<tr>
<td>41</td>
<td>xs:language</td>
<td>atomic value</td>
</tr>
<tr>
<td>42</td>
<td>xs:NMTOKEN</td>
<td>atomic value</td>
</tr>
<tr>
<td>43</td>
<td>xs:Name</td>
<td>atomic value</td>
</tr>
<tr>
<td>44</td>
<td>xs:NCName</td>
<td>atomic value</td>
</tr>
<tr>
<td>45</td>
<td>xs:ID</td>
<td>atomic value</td>
</tr>
<tr>
<td>46</td>
<td>xs:IDREF</td>
<td>atomic value</td>
</tr>
<tr>
<td>47</td>
<td>xs:ENTITY</td>
<td>atomic value</td>
</tr>
<tr>
<td>48</td>
<td>xs:float</td>
<td>atomic value</td>
</tr>
<tr>
<td>49</td>
<td>xs:double</td>
<td>atomic value</td>
</tr>
<tr>
<td>50</td>
<td>xs:decimal</td>
<td>atomic value</td>
</tr>
<tr>
<td>51</td>
<td>xs:precisionDecimal</td>
<td>atomic value</td>
</tr>
<tr>
<td></td>
<td>xs:integer</td>
<td>atomic value</td>
</tr>
<tr>
<td>---</td>
<td>-----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>53</td>
<td>xs:nonPositiveInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>54</td>
<td>xs:negativeInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>55</td>
<td>xs:long</td>
<td>atomic value</td>
</tr>
<tr>
<td>56</td>
<td>xs:int</td>
<td>atomic value</td>
</tr>
<tr>
<td>57</td>
<td>xs:short</td>
<td>atomic value</td>
</tr>
<tr>
<td>58</td>
<td>xs:byte</td>
<td>atomic value</td>
</tr>
<tr>
<td>59</td>
<td>xs:nonNegativeInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>60</td>
<td>xs:unsignedLong</td>
<td>atomic value</td>
</tr>
<tr>
<td>61</td>
<td>xs:unsignedInt</td>
<td>atomic value</td>
</tr>
<tr>
<td>62</td>
<td>xs:unsignedShort</td>
<td>atomic value</td>
</tr>
<tr>
<td>63</td>
<td>xs:unsignedByte</td>
<td>atomic value</td>
</tr>
<tr>
<td>64</td>
<td>xs:positiveInteger</td>
<td>atomic value</td>
</tr>
<tr>
<td>65</td>
<td>xs:duration</td>
<td>atomic value</td>
</tr>
<tr>
<td>66</td>
<td>xs:yearMonthDuration</td>
<td>atomic value</td>
</tr>
<tr>
<td>67</td>
<td>xs:dayTimeDuration</td>
<td>atomic value</td>
</tr>
<tr>
<td>68</td>
<td>xs:dateTime</td>
<td>atomic value</td>
</tr>
<tr>
<td>69</td>
<td>xs:dateTimeStamp</td>
<td>atomic value</td>
</tr>
<tr>
<td>70</td>
<td>xs:date</td>
<td>atomic value</td>
</tr>
<tr>
<td>71</td>
<td>xs:time</td>
<td>atomic value</td>
</tr>
<tr>
<td>72</td>
<td>xs:gYearMonth</td>
<td>atomic value</td>
</tr>
<tr>
<td>73</td>
<td>xs:gYear</td>
<td>atomic value</td>
</tr>
<tr>
<td>74</td>
<td>xs:gMonthDay</td>
<td>atomic value</td>
</tr>
<tr>
<td>75</td>
<td>xs:gDay</td>
<td>atomic value</td>
</tr>
<tr>
<td>76</td>
<td>xs:gMonth</td>
<td>atomic value</td>
</tr>
<tr>
<td>77</td>
<td>xs:boolean</td>
<td>atomic value</td>
</tr>
<tr>
<td>78</td>
<td>base64:binary</td>
<td>atomic value</td>
</tr>
<tr>
<td>79</td>
<td>xs:base64Binary</td>
<td>atomic value</td>
</tr>
<tr>
<td>80</td>
<td>xs:hexBinary</td>
<td>atomic value</td>
</tr>
<tr>
<td>81</td>
<td>xs:anyURI</td>
<td>atomic value</td>
</tr>
<tr>
<td>82</td>
<td>xs:QName</td>
<td>atomic value</td>
</tr>
<tr>
<td>83</td>
<td>xs:NOTATION</td>
<td>atomic value</td>
</tr>
</tbody>
</table>
Chapter 100. Java Examples

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section. The following Java code snippets demonstrate how easy it is to run database commands, create collections, perform queries, etc. by integrating the BaseX code. Most examples are taken from our basex-examples repository, in which you will find some more use cases.

Local Examples

The following code snippets work in embedded mode; they do not rely on an additional server instance:

• RunCommands.java creates and drops database and index instances, prints a list of all existing databases.
• RunQueries.java shows three variants of running queries.
• BindContext.java demonstrates how a value can be bound as context item.
• BindVariables.java demonstrates how a value can be bound to a variable.
• CreateCollection.java creates and manages a collection.
• QueryCollection.java creates, runs queries against it and drops a collection.
• WikiExample.java creates a database from an url (wiki instance), runs a query against it and drops the database.

Server Examples

The examples below take advantage of the client/server architecture:

• ServerCommands.java launches server-side commands using a client session.
• ServerAndLocal.java processes server results locally.
• ServerConcurrency.java runs concurrent queries.
• ServerQueries.java shows how iterative queries can be performed.
• UserExample.java manages database users.

XQuery Module Examples

BaseX provides Java Bindings for accessing external Java code via XQuery functions. The following examples show how this feature can be utilized:

• FruitsExample.java demonstrates how Java classes can be imported as XQuery modules.
• FruitsModule.java is a simple demo module called by FruitsExample.
• ModuleDemo.java is a simple XQuery demo module that demonstrates how XQuery items can be processed from Java. It is derived from the QueryModule class.
• QueryModule.java is located in the BaseX core. Java query modules can extend this class to get access to the current query context and enrich functions with properties ()

XQJ API

The implementation of the BaseX XQJ API (closed-source) has been written by Charles Foster. It uses the client/server architecture. The basex-examples repository contains various examples on how to use XQJ.
Client API

- *BaseXClient.java* provides an implementation of the Server Protocol.
- *Example.java* demonstrates how commands can be executed on a server.
- *QueryExample.java* shows how queries can be executed in an iterative manner.
- *QueryBindExample.java* shows how external variables can be bound to XQuery expressions.
- *CreateExample.java* shows how new databases can be created.
- *AddExample.java* shows how documents can be added to databases, and how existing documents can be replaced.
- *BinaryExample.java* shows how binary resource can be added to and retrieved from the database.

REST API

- *RESTGet.java* presents the HTTP GET method.
- *RESTPost.java* presents the HTTP POST method.
- *RESTPut.java* presents the HTTP PUT method.
- *RESTAll.java* runs all examples at one go.

XML:DB API (deprecated)

Note that the XML:DB API does not talk to the server and can thus only be used in embedded mode.

- *XMLDBCreate.java* creates a collection using XML:DB.
- *XMLDBQuery.java* runs a query using XML:DB.
- *XMLDBInsert.java* inserts a document into a database using XML:DB.
Part X. Extensions
Chapter 101. Docker

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section.

The BaseX server is available as automated build `basex/basexhttp` on the Docker Hub, providing both release and nightly builds. All images are automatically rebuilt if Docker provides updated base images.

**Running a BaseX Container**

To start a BaseX container based on the latest development release publishing the BaseX server and HTTP ports 1984 and 8984 and bind-mounting your user’s data directory, run

```
docker run -ti \
  --name basexhttp \
  --publish 1984:1984 \
  --publish 8984:8984 \
  --volume "$(pwd)/basex/data":/srv/basex/data \
  basex/basexhttp:latest
```

By passing any other BaseX executable, you can also for example run a BaseX client connecting to the linked BaseX server for management operations on the BaseX command line:

```
docker run -ti \
  --link basexhttp:basexhttp \
  basex/basexhttp:latest basexclient -nbasexhttp
```

BaseX is run under the `basex` user with fixed user ID 1984. The user’s home directory is `/srv`. Several ports are exposed:

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1984</td>
<td>Server port</td>
</tr>
<tr>
<td>8984</td>
<td>HTTP port</td>
</tr>
<tr>
<td>8985</td>
<td>HTTP stop port</td>
</tr>
</tbody>
</table>

Leaving BaseX defaults but `--publish`ing them under another external port is recommended if you want to change the ports.

**Security Considerations**

The Docker image ships the unchanged default credentials. Especially if you publish the server port 1984 or link a public DBA instance against the container, make sure to change the default credentials. When publishing ports, consider which interfaces to bind to, paying special attention to the server port.

A common use case will be linking a well-researched and mature reverse proxy link `nginx` against the application container. Goals are to reduce exposure of BaseX and Jetty, adding TLS-encryption, serve static resources like images and perform URL rewrites as needed. If you need to access the command line, you can always `docker exec` into the container and run `basexclient`.

**Running your own Application**

If you want to add your own application in a Docker image, create an image `FROM basex/basexhttp:[tag]` with `[tag]` being the BaseX version you’re developing against. Unless configured otherwise, you will add your application code to `/srv/basex/webapp` and modules to `/srv/basex/repo`. 438
Example: DBA

An example for creating your own Docker image based on basex/basexhttp is the DBA application. A Dockerfile was added to the source code’s root directory. The very simple file contains only few statements:

```
FROM basex/basexhttp:latest
MAINTAINER BaseX Team <basex-talk@mailman.uni-konstanz.de>
COPY . /srv/basex/webapp
```

For general production usage, you should choose a fixed version instead of the development branch behind latest, so your application does not suddenly break because of unnoticed API changes. The most relevant part happens in the `COPY` statement, which adds the file contents to the `webapp` directory. That’s already it -- you’re ready to run.

Advanced Usage

BaseX Configuration

If you need to adjust the BaseX configuration to tune the default options, add a `.basex` file to `/srv`:

```
COPY .basex /srv/basex
```

Options not defined in the `.basex` file will be automatically set to the default values. Users and passwords can be defined by adding a `users.xml` file, which is described on the User Management page.

Jetty Configuration

If you need to change the embedded web server configuration, you can always `COPY` a `WEB-INF` folder containing the required files and overwrite the predefined configuration.

Java Runtime Parameters

Larger applications and databases might require adjusted JRE parameters like increasing the memory limit. You can change those by setting the `BASEX_JVM` environment variable:

```
ENV BASEX_JVM="-Xmx2048m"
```

Installing Debian Packages

The `basex/basexhttp` Docker image is built on the official Maven Docker image `maven:3-jdk-8-alpine`, which in turn derives from alpine. You can add arbitrary packages via APK. Make sure to switch to the root user context before installing packages and back to the `basex` user afterwards. As common in the Docker environment, you need to fetch the package catalog using `apt-get update` before installing packages and should clean up afterwards to keep the image small. This example installs some libraries required for image manipulation and adds them to the `$CLASSPATH`:

```
USER root
RUN apk update && apk add --no-cache git
USER basex
```
Chapter 102. YAJSW

Read this entry online in the BaseX Wiki.

This page is part of the Developer Section.

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. Select the Zip Archive.

• Download the latest version of YAJSW.

• Download the latest version of Java.

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

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)

```java
# 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
wrapper.logfile=${wrapper.working.dir}/data/\logs/\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. At this point you should go ahead and set a password for the admin user.

1. Open a web browser and go to http://localhost:8984/dba (or http://host:8984/dba from your computer, replace 'host' with the address of the server) to open the database administration web console.

2. Log in with username 'admin' password 'admin'

3. Click on Users to navigate to the user management screen.

4. Click on the admin user

5. Set a password for the admin user and then click Save.
Chapter 103. Android

Read this entry online in the BaseX Wiki.

It is possible to create an Android port of BaseX. Therefore 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[1] 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'[2] 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.
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");
        }
    }
}
```

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[3].

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),
```
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[4] 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.

```groovy
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
}
```
Part XI. Advanced User's Guide
Chapter 104. Advanced User's Guide

Read this entry online in the BaseX Wiki.

This page is one of the Main Sections of the documentation. It contains details on the BaseX storage and the Server architecture, and presents some more GUI features.

Storage

- **Configuration**: BaseX start files and directories
- **Backups**: Backup and restore databases
- **Catalog Resolver**: Information on entity resolving
- **Storage Layout**: How data is stored in the database files

Use Cases

- **Statistics**: Exemplary statistics on databases created with BaseX
- **Twitter**: Storing live tweets in BaseX

Server and Query Architecture

- **User Management**: User management in the client/server environment
- **Transaction Management**: Insight into the BaseX transaction management
- **Logging**: Description of the server logs
Chapter 105. Configuration

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User’s Guide. It gives some more insight into the configuration of BaseX.

Configuration Files

BaseX maintains some configuration files, which are stored in the project’s Home Directory:

- `.basex` contains all options that are relevant for running the server or standalone versions of BaseX.
- `.basexgui` defines all options relevant to the BaseX GUI.
- `.basexhistory` contains commands that have been typed in most recently.
- `.basexhome` can be created by a user to mark a folder as home directory. Its contents do not matter, so it is usually empty.

Note that:

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

Home Directory

As BaseX is distributed in different flavors, and 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 chosen 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 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), from 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 what directory was chosen. You can run the XQuery expression `Q{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 sub-directory of the database folder (see Logging for more information).

Changelog

Version 9.0

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

Version 8.0

• Updated: .basexperrm is obsolete. Users are now stored in users.xml in the database directory (see User Management for more information).

Version 7.7

• Updated: The .basehome file marks a folder as home directory.
Chapter 106. Backups

Read this entry online in the BaseX Wiki.

This page is part of the Advanced User's Guide. The following two paragraphs demonstrate how to create a backup and restore the database within BaseX.

GUI Example

1. Start the BaseX GUI and create a new database in Database → New... with your XML document.
2. Go to Database → Manage... and create a backup of your database. The backup will be created in the database directory.
3. Go to Database → Add... and add another document.
4. Go to Database → Manage... and restore your database. The database will be restored from the latest backup of to the database found in the database directory.

Console Example

1. Start the BaseX Standalone client from a console.
2. Create a new database via the CREATE DB command.
3. Use the CREATE BACKUP command to back up your database.
4. Add a new document via ADD: ADD AS newdoc.xml <newdoc/>
5. Use the RESTORE command to restore the original database.
6. Type in XQUERY / to see the restored database contents.

The same commands can be used with a BaseX client connected to a remote Database Server.
Chapter 107. Catalog Resolver

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It clarifies how to deal with external DTD declarations when parsing XML data.

Overview

XML documents often rely on Document Type Definitions (DTDs). While parsing a document with BaseX, entities can be resolved with respect to that particular DTD. By default, the DTD is only used for entity resolution.

XHTML, for example, defines its doctype via the following line:

```xml
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Fetching `xhtml1-strict.dtd` obviously involves network traffic. When dealing with single files, this may seem tolerable, but importing large collections benefits from caching these resources. Depending on the remote server, you will experience significant speed improvements when caching DTDs locally.

XML Entity and URI Resolvers

BaseX comes with a default URI resolver that is usable out of the box.

To enable entity resolving you have to provide a valid XML Catalog file, so that the parser knows where to look for mirrored DTDs.

A simple working example for XHTML might look like this:

```xml
<?xml version="1.0"?>
<catalog prefer="system" xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
  <rewriteSystem systemIdStartString="http://www.w3.org/TR/xhtml1/DTD/" rewritePrefix="file:///path/to/dtds/"/>
</catalog>
```

This rewrites all systemIds starting with: `http://www.w3.org/TR/xhtml1/DTD/` to `file:///path/to/dtds/`.

The XHTML DTD `xhtml1-strict.dtd` and all its linked resources will now be loaded from the specified path.

GUI Mode

When running BaseX in GUI mode, simply provide the path to your XML Catalog file in the Parsing Tab of the Database Creation Dialog.

Console & Server Mode

To enable Entity Resolving in Console Mode, assign a catalog file path to the CATFILE option. All subsequent ADD commands will use the specified catalog file to resolve entities.

The paths to your catalog file and the actual DTDs are either absolute or relative to the current working directory. When using BaseX in Client-Server-Mode, this is relative to the server’s working directory.

Please Note

Entity resolving only works if the internal XML parser is switched off (which is the default case). If you use the internal parser, you can manually specify whether you want to parse DTDs and entities or not.
Using other Resolvers

There might be some cases when you do not want to use the built-in resolver that Java provides by default (via `com.sun.org.apache.xml.internal.resolver.*`).

BaseX offers support for the Apache-maintained XML Commons Resolver, available for download here.

To use it add `resolver.jar` to the classpath when starting BaseX:

```
java -cp basex.jar:resolver.jar org.basex.BaseXServer
```

More Information

- Wikipedia on Document Type Definitions
- Apache XML Commons Article on Entity Resolving
- XML Entity and URI Resolvers, Sun
Chapter 108. Storage Layout

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It presents some low-level details on how data is stored in the database files.

Data Types

The following data types are used for specifying the storage layout:

<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.

Meta Data, Name/Path/Doc Indexes: .inf

<table>
<thead>
<tr>
<th>Description</th>
<th>Format</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Meta Data</td>
<td></td>
<td>DiskData() MetaData() Users()</td>
</tr>
<tr>
<td>2. Main memory indexes</td>
<td></td>
<td>DiskData()</td>
</tr>
<tr>
<td>2 a) Name IndexElement/attribute names</td>
<td></td>
<td>Names() TokenSet.read() StatsKey()</td>
</tr>
</tbody>
</table>
### Storage Layout

<table>
<thead>
<tr>
<th>2 b) Path Index</th>
<th>1. Flag for path definition (Boolean, always true; legacy)</th>
<th>PathSummary() PathNode()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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 discarded 2.6. Recursive generation of child nodes (→ 2)</td>
<td></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) | Namespaces() NSNode() |

| 2 d) Document Index | Array of integers, representing the distances between all document pre values (Nums) | DocIndex() |

#### Node Table: tbl, tbli
- **tbl**: Main database table, stored in blocks.
- **tbli**: Database directory, organizing the database blocks.

Some more information on the node storage is available.

#### Texts: txt, atv
- **txt**: Heap file for text values (document names, string values of texts, comments and processing instructions)
- **atv**: Heap file for attribute values.

#### Value Indexes: txtl, txtr, atvl, atvr

**Text Index:**
- **txtl**: Heap file with ID lists.
- **txtr**: Index file with references to ID lists.

The **Attribute Index** is contained in the files atvl and atvr, 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.

#### Full-Text Fuzzy Index: ftxx, ftxy, ftxz

...may soon be reimplemented.
Chapter 109. 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:

```bash
$ 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:

```bash
$ 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:open-pre and db:open-id can be used to go back and retrieve the original node. By default, id lookups are expensive. If the UPDINDEX option is turned on, an additional index will be maintained to speed up the process.

**Block Storage**

BaseX logically splits the tbl.base 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.base) 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.base, too.

Since these two maps will not grow excessively large, but are accessed resp. changed on each read resp. write operation, they are kept in main memory and flushed to disk on closing the database.
A newly created database with 256 + 10 records will occupy the first two blocks with physical addresses 0 and 4096. The corresponding fpre's will be 0 and 256.

If a record with pre = 12 is to be inserted, it needs to be stored in the first block, which is, however, full. In this case, a new block with physical address 8192 will be allocated, the records with pre values from 12 to 255 will be copied to the new block, the new record will be stored in the old block at pre = 12, and the two maps will look like this:

| fpre's     | 0, 13, 257 |
| addr's     | 0, 8192, 4096 |

Basically, the old records remain in the first block, but they will not be read, since the fpre's array says that only 13 records are stored in the first block. This causes redundant storage of the records with old pre's from 13 to 255.

Additionally to these two maps (fpre's and addr's), BaseX maintains a bit map (which is also stored in tbli.basex) which reflects which physical blocks are free and which not, so that when a new block is needed, an already free one will be reused.
Chapter 110. User Management

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. The user management defines which permissions are required by a user to perform a database command or XQuery expression.

Permissions are mostly relevant in the client/server architecture, as the Standalone Mode and the GUI is run with admin permissions. There are a few exceptions such as the `xquery:eval` function: Its execution scope can also be limited by specifying a permission.

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.

![Permissions Hierarchy Diagram]

Permissions hierarchy 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
```
User Management

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:

db: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 the AUTHMETHOD is set to Digest.

Changelog

Revised in Version 8.0.
Chapter 111. 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. So it is advisable to use the BACKUP command to regularly backup 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.
XQuery Locks

By default, access to external resources (files on hard disk, HTTP requests, ...) is not controlled by the transaction monitor of BaseX. You can use custom XQuery locks to do so:

Options, Pragmas, Annotations

*Introduced with Version 9.1:* locks via pragmas and function annotations.

- You can declare custom read and write locks via options, pragmas or function annotations.
- The value of the lock may contain one or multiple lock keys (separated with commas). The default value is an empty string.
- Similar to the internal database locks, write locks block all other operations while read locks allow parallel access.
- The internal locks and XQuery locks can co-exist (there will be no conflicts, even if your lock string equals the name of a database that will be locked by the transaction manager).

In the following module, lock annotations are used to prevent concurrent write operations on the same file:

```xml
module namespace config = 'config';
declare %basex:read-lock('CONFIG') function config:read() {
    file:read-text('config.txt')
};
declare %basex:write-lock('CONFIG') function config:write($data) {
    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 on this lock string will be set for this query.
- If a query calls `config:write`, it will be queued until there is no running query left that has `CONFIG` locked.
- If the writing query will be evaluated, all other queries that will set a `CONFIG` lock (reading or writing) will have to wait.

Local locks can also be declared via pragmas:

```xml
(# basex:write-lock CONFIG #) {
    file:write('config.xml', <config/>)
}
```

Locks for the functions of a module can be assigned via option declarations:

```xml
declare option basex:write-lock 'CONFIG';
file:write('config.xml', <config/>)
```

Before Version 9.1, locks were declared in the `query` namespace.

Java Modules

Locks can also be acquired on Java functions which are imported and invoked from an XQuery expression. It is advisable to explicitly lock Java code whenever it performs sensitive read and write operations.
Limitations

Commands

Database locking works with all commands unless the glob syntax is used, such as in the following command call:

- **DROP DB new*: drop all databases starting with "new"

XQuery

Deciding which databases will be accessed by a complex XQuery expression is a non-trivial task. Database detection works for the following types of queries:

- `//item`, read-locking of the database opened by a client
- `doc('factbook')`, read-locking of "factbook"
- `collection('db/path/to/docs')`, read-locking of "db"
- `fn:sum(1 to 100)`, locking nothing at all
- `delete nodes doc('test')///*[string-length(local-name(.)) > 5]`, write-locking of "test"

All databases will be locked by queries of the following kind:

- `for $db in ('db1', 'db2') return doc($db)`
- `doc(doc('test')/reference/text())`
- `let $db := 'test' return insert nodes <test/> into doc($db)`

You can consult the query info output (which you find in the Info View of the GUI or which you can turn on by setting QUERYINFO to true) to find out which databases have been locked by a query.

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 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 112. 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.

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.

RESTXQ

Non-trivial web applications provide the ability for users to sign in and out. User handling can be realized with session attributes (see e.g. the DBA code).

By default, RESTXQ code is run with admin permissions. However, as it is more interesting to know which user has called a function, the RESTXQ user string (which is admin by default, and which will show up in the log data) will be overwritten by the value of an id session attribute. If the request path includes /dba/, the dba session attribute will be assigned.

If the following function is called more than once, joe will appear as user in the second and subsequent log entries:

import module namespace Session = 'http://basex.org/modules/session';
declare %rest:path('/session-id') function local:f() {
    Session:set('id', 'joe'),
    'I am Joe'
};

Format

Example 1

<table>
<thead>
<tr>
<th>Time</th>
<th>User</th>
<th>Request Type</th>
<th>Status</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:18:12.892</td>
<td>SERVER</td>
<td>admin</td>
<td>OK</td>
<td>Server was started (port: 1984)</td>
</tr>
<tr>
<td>01:18:15.436</td>
<td>127.0.0.1:4722</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>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></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

463
A user john has performed an iterative query, using one of the client APIs.

Example 3

An admin user has accessed the factbook database via REST.

Changelog

Version 8.6

- Added: The log directory can be changed with the LOGPATH option.
- Updated: Include session attributes in log data.
Part XII. Use Cases
## Chapter 113. 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 GiB</td>
<td>2'147'483'648 MiB</td>
<td>32768</td>
<td>12^2</td>
<td>2^256</td>
<td>2^256</td>
<td>no limit</td>
<td></td>
</tr>
</tbody>
</table>

<p>| RuWikiHist | 421 GiB | 1 | 416 GiB | 324'848'508 | 21 | 6 | 2 |
| ZhWikiHist | 26 GiB | 1 | 120 GiB | 179'199'662 | 21 | 6 | 2 |
| EnWikiHist | 55 GiB | 1 | 75 GiB | 134'380'393 | 21 | 6 | 2 |
| XMark     | 55 GiB | 1 | 64 GiB | 1'651'071'348 | 74 | 9 | 0 |
| EnWikiMeta| 56 GiB | 1 | 52 GiB | 401'456'348 | 21 | 6 | 2 |
| MedLine   | 38 GiB | 1 | 37 GiB | 1'623'764'254 | 84 | 6 | 0 |
| iProClass | 36 GiB | 1 | 37 GiB | 1'631'218'984 | 245 | 4 | 2 |
| Inex2009  | 31 GiB | 1 | 34 GiB | 1'336'110'639 | 28'034 | 451 | 1 |
| CoPhIR    | 29 GiB | 1 | 25 GiB | 198'546'747 | 24 | 21 | 2 |
| XMark     | 22 GiB | 1 | 26 GiB | 645'999'762 | 74 | 0 | 9 |
| InterPro  | 14 GiB | 1 | 19 GiB | 860'304'235 | 7 | 15 | 0 |
| Genome1   | 13 GiB | 1 | 13 GiB | 432'628'103 | 26 | 101 | 2 |
| NewYork1  | 13 GiB | 1 | 13 GiB | 280'407'005 | 41 | 33 | 0 |
| TrEMBL    | 11 GiB | 1 | 14 GiB | 589'650'538 | 47 | 30 | 2 |
| XMark     | 11 GiB | 1 | 13 GiB | 323'083'402 | 74 | 9 | 0 |
| IntAct    | 7973 MiB | 25'624 | 6717 MiB | 297'478'397 | 64 | 22 | 2 |
| Freebase  | 7366 MiB | 1 | 10 GiB | 443'627'994 | 61 | 283 | 1 |
| SDMX      | 6356 MiB | 1 | 8028 MiB | 395'871'872 | 22 | 6 | 3 |
| OpenStreet | 25'624 MiB | 1 | 5171 MiB | 6'910'669 | 19 | 5 | 2 |
| SwissProt | 4604 MiB | 1 | 5422 MiB | 241'274'408 | 70 | 39 | 2 |
| EURLex    | 4815 MiB | 1 | 5532 MiB | 167'328'032 | 186 | 46 | 1 |
| Wikicorpus| 444 MiB | 1 | 443 MiB | 157'948'562 | 125'279 | 2'687 | 2 |
| EnWikiRD  | 8679 MiB | 1 | 3537 MiB | 98'433'194 | 11 | 2 | 11 |
| CoPhIR    | 2695 MiB | 1 | 2700 MiB | 101'638'851 | 42 | 42 | 0 |
| MeSH      | 2091 MiB | 1 | 2410 MiB | 104'845'819 | 6 | 5 | 2 |
| FreeDB    | 1723 MiB | 1 | 2462 MiB | 102'901'512 | 7 | 3 | 0 |
| XMark     | 1134 MiB | 1 | 1303 MiB | 32'298'989 | 74 | 9 | 0 |</p>
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Files</th>
<th>#Files</th>
<th>#DbSize</th>
<th>#Nodes</th>
<th>#Attr</th>
<th>#ENames</th>
<th>#ANames</th>
<th>#URIs</th>
</tr>
</thead>
<tbody>
<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>1'777'495</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></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>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LibraryUK</td>
<td>760 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>1'777'495</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></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>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LibraryUK</td>
<td>760 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>1'777'495</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></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>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LibraryUK</td>
<td>760 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>1'777'495</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></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>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>LibraryUK</td>
<td>760 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>1'777'495</td>
<td>767 MiB</td>
<td>15'309'0150</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Organizations</td>
<td>833 MiB</td>
<td>1</td>
<td>724 MiB</td>
<td>33'112'3923</td>
<td>38</td>
<td>9</td>
<td>0</td>
<td>7</td>
<td></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>604 MiB</td>
<td>5'933'713</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</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/airbasexml">http://air-climate.eionet.europa.eu/databases/airbase/airbasexml</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/">http://www.xml-benchmark.org/</a> generated 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>Source</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MusicXML</td>
<td><a href="http://www.recordare.com/xml/samples.html">http://www.recordare.com/xml/samples.html</a></td>
</tr>
<tr>
<td>Nasa</td>
<td><a href="http://www.cs.washington.edu/research/xmldatasets/repository.html">http://www.cs.washington.edu/research/xmldatasets/repository.html</a></td>
</tr>
<tr>
<td>OpenStreetMap</td>
<td><a href="http://dump.wiki.openstreetmap.org/osmwiki-latest-files.tar.gz">http://dump.wiki.openstreetmap.org/osmwiki-latest-files.tar.gz</a></td>
</tr>
<tr>
<td>Organizations</td>
<td><a href="http://www.data.gov/raw/1358">http://www.data.gov/raw/1358</a></td>
</tr>
<tr>
<td>RuWikiHist</td>
<td><a href="http://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-meta-history.xml.7z">http://dumps.wikimedia.org/ruwiki/latest/ruwiki-latest-pages-meta-history.xml.7z</a></td>
</tr>
<tr>
<td>SDMX</td>
<td><a href="http://www.metadatatechnology.com/">http://www.metadatatechnology.com/</a></td>
</tr>
<tr>
<td>Shakespeare</td>
<td><a href="http://www.cafeconleche.org/examples/shakespeare">http://www.cafeconleche.org/examples/shakespeare</a></td>
</tr>
<tr>
<td>Thesaurus</td>
<td><a href="http://www.drze.de/BELIT/thesaurus">http://www.drze.de/BELIT/thesaurus</a></td>
</tr>
<tr>
<td>Treebank</td>
<td><a href="http://www.cs.washington.edu/research/xmldatasets">http://www.cs.washington.edu/research/xmldatasets</a></td>
</tr>
<tr>
<td>TreeOfLife</td>
<td><a href="http://tolweb.org/data/tolskeletaldump.xml">http://tolweb.org/data/tolskeletaldump.xml</a></td>
</tr>
<tr>
<td>Wikicorpus</td>
<td><a href="http://www-connex.lip6.fr/~denoyer/wikipediaXML">http://www-connex.lip6.fr/~denoyer/wikipediaXML</a></td>
</tr>
<tr>
<td>XMark</td>
<td><a href="http://www.xml-benchmark.org/">http://www.xml-benchmark.org/</a> generated with xmlgen</td>
</tr>
<tr>
<td>ZDNET</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>ZhWikiHist</td>
<td><a href="http://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-meta-history.xml.7z">http://dumps.wikimedia.org/zhwiki/latest/zhwiki-latest-pages-meta-history.xml.7z</a></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 114. Twitter

Read this entry online in the BaseX Wiki.

This article is part of the Advanced User's Guide. It is about the usage of BaseX for processing and storing the live data stream of Twitter. We illustrate some statistics about the Twitter data and the performance of BaseX.

As Twitter attracts more and more users (over 140 million active users in 2012) and is generating large amounts of data (over 340 millions of short messages ('tweets') daily), it became a really exciting data source for all kind of analytics. Twitter provides the developer community with a set of APIs for retrieving the data about its users and their communication, including the Streaming API for data-intensive applications, the Search API for querying and filtering the messaging content, and the REST API for accessing the core primitives of the Twitter platform.

BaseX as Twitter Storage

For retrieving the Twitter stream we connect with the Streaming API to the endpoint of Twitter and receive a never ending tweet stream. As Twitter delivers the tweets as JSON objects the objects has to be converted into XML fragments. For this purpose the parse function of the XQuery JSON Module is used. In the examples section both versions are shown (tweet as JSON and tweet as XML). For storing the tweets including the meta-data, we use the standard insert function of XQuery Update.

Twitter’s Streaming Data

Each tweet object in the data stream contains the tweet message itself and over 60 data fields (for further information see the fields description). The following section shows the amount of data, that is delivered by the Twitter Streaming API to the connected endpoints with the 10% gardenhose access per hour on the 6th of the months February, March, April and May. It is the pure public live stream without any filtering applied.

Statistics

<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>Fri, 6-Apr-2012</td>
<td>Total tweets</td>
<td>34.638.976</td>
</tr>
<tr>
<td></td>
<td>Average tweets per hour</td>
<td>1.443.290</td>
</tr>
</tbody>
</table>
Average tweets per minute 24.054
Average tweets per second 400
Sun, 6-May-2012 Total tweets 35.982.976
Average tweets per hour 1.499.290
Average tweets per minute 24.988
Average tweets per second 416

Example Tweet (JSON)

{
   "contributors": null,
   "text": "Using BaseX for storing the Twitter Stream",
   "geo": null,
   "retweeted": false,
   "in_reply_to_screen_name": null,
   "possibly_sensitive": false,
   "truncated": false,
   "entities": {
      "urls": [ ],
      "hashtags": [ ],
      "user_mentions": [ ]
   },
   "in_reply_to_status_id_str": null,
   "id": 1984009055807******,
   "in_reply_to_user_id_str": null,
   "source": "<a href="http://\twitterfeed.com" rel="nofollow">twitterfeed</a>",
   "favorited": false,
   "in_reply_to_status_id": null,
   "retweet_count": 0,
   "created_at": "Fri May 04 13:17:16 +0000 2012",
   "in_reply_to_user_id": null,
   "possibly_sensitive_editable": true,
   "id_str": "1984009055807******",
   "place": null,
   "user": {
      "location": "",
      "default_profile": true,
      "statuses_count": 9096,
      "profile_background_tile": false,
      "lang": "en",
      "profile_link_color": "0084B4",
      "id": 5024566**,
      "following": null,
      "protected": false,
      "favourites_count": 0,
      "profile_text_color": "333333",
      "contributors_enabled": false,
      "verified": false,
      "description": "http://\basex.org",
      "profile_sidebar_border_color": "CODEED",
      "name": "BaseX",
      "profile_background_color": "CODEED",
      "created_at": "Sat Feb 25 04:05:30 +0000 2012",
      "default_profile_image": true,
      "followers_count": 860,
   },
Example Tweet (XML)

```xml
<json booleans="retweeted possibly__sensitive truncated favorited possibly__sensitive__editable default__profile profile__background__tile protected contributors__enabled verified default__profile__image geo__enabled profile__use__background__image show_all__inline__media is__translator"
  numbers="id retweet__count statuses__count favourites__count followers__count friends__count listed_count"
nulls="contributors geo in__reply__to__screen__name
  in__reply__to__status__id__str in__reply__to__user__id__str
  in__reply__to__status__id in__reply__to__user__id place following
  follow_request_sent utc_offset time__zone notifications coordinates"
arrays="urls indices hashtags user__mentions"
objects="json entities user">
<contributors/>
<text>Using BaseX for storing the Twitter Stream</text>
<geo/>
<retweeted>false</retweeted>
<in__reply__to__screen__name/>
<possibly__sensitive>false</possibly__sensitive>
<truncated>false</truncated>
<entities>
  <urls/>
  <user__mentions/>
</entities>
<in__reply__to__status__id__str/>
<id>1984009055807*****</id>
<in__reply__to__user__id__str/>
<source><a href="http://twitterfeed.com" rel="nofollow">twitterfeed</a></source>
<favorited>false</favorited>
<in__reply__to__status__id/>
<tweet__count>0</tweet__count>
<created_at>Fri May 04 13:17:16 +0000 2012</created_at>
<is__translator>true</is__translator>
<in__reply__to__user__id__str/>
<possibly__sensitive__editable>true</possibly__sensitive__editable>
<id__str>1984009055807*****</id__str>
</json>
```
BaseX Performance

The test show the time BaseX needs to insert large amounts of real tweets into a database. We can derive that BaseX scales very well and can keep up with the incoming amount of tweets in the stream. Some lower values can occur, cause the size of the tweets differ according to the meta-data contained in the tweet object. Note: The AUTOFLUSH option is set to FALSE (default: SET AUTOFLUSH TRUE)

System Setup: Mac OS X 10.6.8, 3.2 GHz Intel Core i3, 8 GB 1333 MHz DDR3 RAM BaseX Version: BaseX 7.3 beta

Insert with XQuery Update

These tests show the performance of BaseX performing inserts with XQuery Update as single updates per tweet or bulk updates with different amount of tweets. The initial database just contained a root node <tweets/> and all incoming tweets are inserted after converting from JSON to XML into the root node. The time needed for the inserts includes the conversion time.
## Single Updates

<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)