

EMC® Documentum® xDB

Version 10.5

Manual

EMC Corporation
Corporate Headquarters:
Hopkinton, MA 01748 9103
1 508 435 1000
www.EMC.com

Copyright © 2000-2013 EMC Corporation. All rights reserved.

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com. Adobe and Adobe PDF Library are trademarks or registered trademarks of Adobe Systems Inc, in the U.S. and other countries. All other trademarks used herein are the property of their respective owners.

Documentation Feedback

Your opinion matters. We want to hear from you regarding our product documentation. If you have feedback about how we can make our documentation better or easier to use, please send us your feedback directly at IIGDocumentationFeedback@emc.com.

Table of Contents

Chapter 1	Quick Start	35
	Getting a quick start with xDB	35
Chapter 2	Introduction.....	37
	xDB Overview	37
	General features	37
	Linking documents with XLink	40
	Versioning and branching.....	40
	Administration tools	41
	Logical architecture	42
	Superuser	42
	Transaction log files	43
	Database objects	43
	Internal structure: databases, segments, files and pages	45
	Database files.....	46
	Database configurations.....	46
	Detachable libraries.....	46
	Managing users and groups.....	47
Chapter 3	Installing xDB.....	49
	Pre-installation requirements.....	49
	Installing xDB on a Windows platform	50
	Upgrading xDB on Windows	57
	Installing xDB on a UNIX platform.....	59
	Upgrading xDB on UNIX	61
	Uninstalling xDB.....	61
	Verifying the xDB installation	61
	Creating a sample database	62
Chapter 4	Configuring xDB.....	65
	Configuration files for Windows	68
	xDB JAR files.....	68
	Using the xhive.bootstrap property	69
	The xDB dedicated page server	70
	Running a background server process on UNIX.....	70
	Running without a dedicated server.....	71
	Using external editors with FTP	71
	Managing DTDs.....	71
	Troubleshooting DTDs.....	72

	Enabling FIPS 140-2 Level 1 Encryption	73
Chapter 5	Optimizing Performance.....	75
	Improving server performance.....	75
	Configuring JVM and cache pages	75
	Choosing the database page size.....	77
	Linux file system performance.....	77
	Using multiple disks.....	77
	Disabling disk-write caches	78
	RPC tracing	78
	Enabling or disabling RPC tracing	79
	Enabling RPC tracing at system level	80
	Sending RPC trace output to console or file.....	80
	Methods for RPC tracing	81
	RPC Trace XML schema example.....	81
Chapter 6	Creating Applications.....	83
	Building and running applications	83
	Running a sample	84
	Creating a database using the API	84
	Connecting to a database	85
	Getting a database configuration.....	86
	Using sessions and transactions	86
	Creating libraries	88
	Storing BLOBs	89
	API methods for managing users and groups.....	90
	Using a RAM segment for temporary data.....	91
	The xDB dedicated page server program	92
	Using the FederationSet API.....	92
	Using xDB with Maven 2.....	94
	Using xDB with Spring	94
	xDB and OSGi	96
	Using xDB with JAAS	97
	Using the API with SSL.....	97
Chapter 7	Managing Documents in Applications.....	99
	Creating and managing documents	99
	Parsing XML documents.....	99
	Parse with context.....	100
	Validating XML documents.....	101
	Normalizing XML documents	102
	Storing XML documents.....	103
	DOM configuration settings.....	103
	Importing non-XML data	106

	Creating a document	108
	Retrieving documents and document parts.....	109
	Using DOM operations	110
	Using document ID	111
	Using document name	111
	Using XQuery	112
	Using indexes.....	112
	Using XPointer with library path	112
	Using XPath and XPointer	114
	Traversing XML documents.....	116
	Using DOM traversal.....	116
	Traversing using function objects	119
	Exporting XML documents	120
	Publishing XML documents.....	121
	XLink interfaces	122
	Using versioning.....	123
	Working with versioned documents.....	123
	Retrieving previous document versions	124
	Branching methods	124
	Node-level versioning.....	125
	Using searchable versions	125
	Using metadata on library children.....	126
	Using abstract schemas.....	126
	Using the APIs for XSL transformations	127
Chapter 8	Session and Transaction Management.....	133
	Sessions, transactions and locking	133
	Namebase and locking	134
	Session lifecycle.....	134
	Joined sessions and session pools.....	136
	Sessions and references to database objects	140
	Referencing database objects in sessions.....	144
	Multithreaded session handling	144
	Transaction isolation in sessions	146
	Managing locking conflicts	146
	Read-only transactions	148
	Getting info on sessions and locks.....	148
Chapter 9	Managing Indexes	149
	Indexes	150
	Index APIs and samples	150
	Path indexes.....	151
	Multipath indexes	153
	Multipath index examples	153
	Differences between multipath and path indexes.....	156
	Multipath index limitations.....	160

Multipath indexing methods.....	160
Customizing the score of multipath indexes	161
Value indexes	161
Using value types for value indexes	161
Value indexing methods	162
Full-text indexes	163
Full-text indexing methods	164
Metadata value indexes	164
Metadata full text indexes	164
Library indexes.....	165
Library indexing methods	165
ID attribute indexes	166
ID attribute indexing methods.....	166
Element name indexes	167
Element name indexing methods.....	167
Concurrent indexes	167
Concurrent indexing methods.....	168
Non-blocking incremental indexes	168
Context conditioned indexes	168
Context conditioned indexing methods.....	169
Optimizing index performance	170
Indexes and timezones.....	171
Chapter 10 XQuery	173
Working with XQueries	173
Working with XQuery methods	173
External XQuery variables and functions.....	175
Accessing documents and libraries with XQuery.....	178
XQuery error reporting.....	178
XQuery options and extension expressions.....	179
XQuery extension functions	182
Using XQuery extension function xhive:force	188
Using XQuery extension function xhive:highlight	188
Using indexes in XQuery	188
Value and name element indexes.....	189
Range queries	191
Indexing metadata	191
Multiple indexes.....	191
Indexes and order by.....	192
Proprietary XQuery extension to order by.....	193
Using type information in XQuery	194

	XQuery full-text search	194
	Full-text search limitations	195
	Full-text logic operators	195
	Queries with wildcards	195
	Queries with fuzzy search	196
	Queries with thesaurus	196
	Queries with thesaurus handler	196
	Anyall options	197
	Positional filters	197
	Cardinality option	198
	Score variables	198
	Score calculation	199
	Boost scoring models	199
	Using boost scoring models	200
	Using the xhive:fts full-text search function	201
	XQuery performance tuning	204
	XQuery collation support	206
	XQuery Profiler	206
	XQuery profiling methods	207
	XQuery implementation	207
	XQuery Security Features	209
	XQuery security methods	209
	XQuery modules	209
	XQuery XML Schema support	210
	XQuery Update Syntax	211
	Proprietary XQuery Update Syntax	211
	Data model differences	213
	Additional XQuery namespace declarations	213
Chapter 11	More methods for XQuery	215
	Use of type information in XQuery	215
	Parallel queries	215
	Using the XQuery Resolver	217
	Preparing XQueries	217
	Extending XQuery using Java	218
	Java objects and instance methods	218
	Type checking	219
	Limitations	219
Chapter 12	Catalogs and Validation	221
	xDB catalogs	221
	Adding models to a catalog	222
	Linking models to documents	222
	Validated parsing	223
	Catalog methods	224
	Validating documents against models	224
	Post Validation Schema Infoset (PSVI)	224
	Accessing PSVI information	225

Chapter 13	Administering xDB	227
	Admin Client	227
	Using the xDB Admin Client.....	228
	Creating a database using the Admin Client.....	230
	Database configuration files.....	230
	Changing the superuser password using the Admin Client.....	232
	Changing the administrator password using the Admin Client	232
	Changing a user password using the Admin Client.....	233
	Importing data	234
	Exporting data	235
	Backing up a federation.....	235
	Restoring a federation backup	236
	Serializing data	237
	Deserializing data	238
	Deserializing the root library.....	238
	Serializing users and groups.....	239
	Deserializing users and groups	239
	Comparison of online backup and serialization.....	239
	Editing documents	240
	Adding indexes.....	241
	Running queries.....	241
	Profiling XQueries.....	243
	Web client.....	246
	Using the command-line client.....	246
	Command-line client commands	248
	Creating a federation.....	250
	Creating a database using the command line.....	251
	The xdb info command.....	251
	Command-line client global options.....	251
	Creating and restoring backups.....	262
	Running incremental backups.....	263
	The xdb backup command.....	263
	The xdb restore command.....	264
	Restoring lost data from log files	265
	Viewing backup metadata.....	266
	Offline backups.....	266
	Suspending xDB activity for snapshot backups	266
	Backing up and restoring a library	267
	Commands for backing up and restoring a library.....	268
	Methods for creating and restoring backups.....	268
	Managing detachable libraries.....	272
	Moving a detachable library	274
	Unusable detachable libraries	275
	Duplicated transaction log files.....	275
	Methods for transaction log duplication.....	276
	Monitoring statistics.....	277
	Enabling statistics monitoring.....	278
	Consuming monitored xDB statistics	278
	Monitored statistics categories	279
	RAM segments	281
	Read-only federations.....	281

	Federation sets	281
	Creating a federation set	282
	Using federation sets.....	282
	Using Secure Sockets Layer (SSL).....	283
	Checking database consistency	283
	Methods for consistency checking.....	284
	Message logging	286
	Message logging areas.....	287
	Message logging framework	287
Chapter 14	Replicating Federations	289
	Replication.....	289
	Creating a federation replica	290
	Running a replicator on a dedicated server	291
	Replication of federation metadata	291
	Changing a replica into a master	292
	Removing a replica.....	292
	Methods for replication	293
	Methods for using a federation replica	293
	Running a replicator on an internal server.....	293
	Read-only transactions with temporary data.....	293
	Using a replica as a failover	294
	Preparing for replication	294
	Replication application code sample.....	295
Chapter 15	Configuring Multiple Backend Servers.....	297
	xDB multi-node architecture	297
	Transaction recovery	299
	Multi-node bootstrap configuration.....	299
	Server upgrade from older xDB release.....	301
	Multi-node considerations	302
	Multi-node run-time restrictions	303
	Managing nodes.....	303
	Methods for multi-node deployment.....	304
	Methods for managing nodes.....	304
	Locking rules	305
	Disabling distributed deadlock detection	305
	Modifying library bindings	305
	Applications for multiple-node configurations	306
	Multiple-node API examples	310
	Replacing a server	314
Chapter 16	Ant Tasks	317
	Using xDB Ant tasks.....	317
	Using the xhive.bootstrap property with Ant.....	318
	xDB Ant type reference.....	318
	Referencing xDB Ant types	322
	xDB Ant task reference.....	323

xDB Documentation

This xDB Manual is designed to provide a technical introduction to the EMC Documentum xDB product. It contains basic xDB information, and discusses installing, configuring, administering and using xDB for software development. There is also more detailed information on more advanced subjects, on specific aspects of xDB, and on using xDB in combination with other tools.

The xDB distribution includes further information for developers, including API documentation and sample code.

Intended audience

This manual is for xDB developers and administrators. It assumes that the reader is familiar with:

- Java and XML
- the operating system that is used with xDB
- basic database principles such as transactions, locking, and access rights
- general principles of client/server architecture and networking

Some knowledge of DOM, XQuery, and XSLT is helpful but not required.

Resources

The [XML Technologies](#) section of the EMC Developer Network offers resources related to xDB, information about specific aspects of xDB and its use, case studies, tools and sample code that you can download.

Support information

EMC Documentum technical support services are designed to make deployment and management of Documentum products as effective as possible.

For the latest product documentation and support materials, including White Papers and Technical Advisories, refer to EMC Online Support (<https://support.emc.com>). Check regularly for new and updated documentation to ensure that you have the latest system information.

Note: Documentation installed, or packaged with the product on the download center, is current at the time of release. Documentation updates made after a release are available for download from EMC Online Support (<https://support.emc.com>).

Typographic conventions

The following table describes the typographic conventions used in this guide.

Table 1 Typographic conventions

Appearance	Meaning
<i>Fixed terms</i>	Terminology, such as names of features, standards, etc.
User interface control	User interface controls, such as menu entries, buttons, etc.
API names	Application Programming Interface elements, such as classes and methods in Java APIs.
Monospaced text	Example code, parameter values.
Commands	Commands and their arguments, to be entered in a command prompt.
File paths	Paths to files in the file system, usually relative to the installation directory.
<i>Variable Names</i>	Variables in command strings and user input variables

Revision History

Changes in release 10.5

- Added a system property `PROP_TRACING_ON=true/false`, to make it possible to disable RPC tracing completely.
- The `xhive.jar` has been made OSGi compatible. In addition, `xhive-api.jar` and `xhive-impl.jar` OSGi bundles have been introduced for use with OSGi Declarative Services; see the updated OSGi sample application.
- `xDB` now requires `xml-apis.jar` as a core dependency.
- Introduced `XhiveDriverFactory.getBackupDriver()` API for obtaining a driver that operates on a federation backup (read-only federation mode).
- Added log output for startup recovery.
- Added support for switching off transactional logging on the library level.
- Added `XhiveBackupInfoIf.getBackupLSNs()` which returns the backup LSN of each and every node involved in the backup.

- Fixed a concurrency issue with the hot backup functionality which prevented users from connecting to the server when the backup was in progress.
- Fixed a bug that causes the backup LSN returned by `XhiveBackupInfo.getBackupLSN()` to be 0.
- Made the API specification of `XhiveBackupInfoIf.getBackupLSNs()` clearer about the return value of `XhiveBackupInfoIf.getBackupLSN()`.
- Fixed the problem that the bootstrap log record generated for `XhiveLibraryIf.removeBinding(String)` is wrong.
- Fixed a recovery issue which might happen if the system crashes at the moment of lucene segment shrinking.
- Fixed a recovery issue which might cause bootstrap file inconsistency if the system crashes.
- Fixed an issue with read-only federations that caused a NPE when obtaining segment information for detachable libraries.
- Fixed an issue with read-only federations where an attempt would be made to modify temporary segments.
- Fixed an issue with read-only federations where an attempt would be made to update the bootstrap file.
- Fixed a bug which in rare cases could cause a federation backup to hang.
- Fixed a deadlock which occurs when a user thread and a system both tried to kill the server.
- Made `XhiveLibraryIf.attach(String, String, String, XhiveFederationFactoryIf.SegmentIdMapper)` official.
- The startup logic will now try to fix violations of data file naming convention.
- **Indexes:** Deprecated `com.xhive.index.interfaces.TokenMetadata` and introduced a custom xDB Lucene attribute `com.xhive.index.interfaces.XhiveWeightAttribute` to be used instead.
- **Indexes:** Fixed an issue where non-Administrator could not create/delete multipath indexes.
- **Indexes:** Fixed an issue where multipath indexes ignored date-times with timezone.
- **Indexes:** Support for unique keys option in the concurrent B-tree indexes.
- **Indexes:** Support for concurrent multipath indexes.
- **Indexes:** Support for intra-collection parallelism in multipath indexes.
- **Indexes:** Support by `IndexInConstructionList` and `IndexInConstruction` of all of the index types, except for `LibraryID` and `LibraryName`.
- **Indexes:** Support for concurrent indexing session of `IndexInConstruction`.
- **Indexes:** Added `moveToIndexList(threshold)` method to `IndexInConstructionListIf`, which attempts to index a number of nodes before moving the index.
- **XQuery:** Updated to the XPath and XQuery Functions and Operators 3.0 spec of 21 May 2013. **Note:** All functions that operate on function items as their arguments have changed. Functions `fn:map` and `fn:map-pairs` are renamed to respectively `fn:for-each` and `fn:for-each-pair`. In addition, the signatures of `fn:filter`, `fn:fold-left` and `fn:fold-right` have changed. As a result, the old functions `fn:map`, `fn:map-pairs`, `fn:filter`, `fn:fold-left` and `fn:fold-right` as defined in the previous spec version (08 January 2013) are no longer supported.
- **XQuery:** Support for XQuery 3.0 Decimal Format Declarations and `fn:format-number`.
- **XQuery:** Support for XQuery Copy-Namespaces declaration.
- **XQuery:** Support for parallel execution of `fn:for-each`.
- **XQuery:** Added extension function `xhive:get-metadata-keys` to get the metadata keys of one or more documents.

- **XQuery:** Changed the default implicit timezone from local time to PT0H. **Note:** WARNING: this change may lead to different results for XQuery functions that depend on the implicit timezone, like `fn:current-dateTime`, `fn:adjust-dateTime-to-timezone` and `fn:implicit-timezone`. Refer to the manual sections on option `xhive:implicit-timezone` and 'Indexes and timezones' for more information on the subject.
- **XQuery:** Fixed an issue where comparison of 2 `dateTimes` was not correct if only one had a timezone.
- **XQuery:** Fixed a multi-path index bug causing wrong score to be calculated in XQueries that used score but did not order by score.
- **XQuery:** Fixed a bug where usage of a for clause with 'allowing empty' could throw an `ArrayIndexOutOfBoundsException`.
- **XQuery:** fixed 5 xquery pretty printer issues. Some of these issues could make an xquery unparsable.
- **XQuery:** Added documentation of the `xhive:return-blobs` XQuery option to the manual.
- **XQuery:** Fixed an issue where a user defined function throwing an `fn:error` caused another exception without the original `fn:error` message.
- **XQuery:** Fixed namespace support in the `xhive:index-paths-values` XQuery option.
- **XQuery:** Fixed a bug where `fn:doc-available` on an empty library caused a NPE.
- **XQuery:** Added interface `XhiveXQueryParallelJobIf` to access sub-query info during parallel query execution.
- **XQuery:** Added extension function `xhive:version-id($doc)` to be used in conjunction with `xhive:collection-*-date()` functions.
- **Admin client, command line client and ant:** added support to access a federation through a federation set (description file or server). The path to such a federation contains a hash. The part before the hash should point to the set, the part after the hash should navigate to the federation. When creating a federation bootstrap file, its path should not contain a hash.
- **Command line client and Ant:** Added new commands (`add-file`, `set-file-maxsize`) and new Ant tasks (`<addsegmentfile/>`, `<setmaxfilesize>`). Added a check so that the maxsize of a datafile (if specified) has to be at least 10 pages.
- **Admin client:** Now able to parse IPv6 addresses.
- **Admin client:** No longer lists reserved segments, as it cannot display information for non-existent data files.
- **Admin client:** Using Check out/ Edit/ Checkin now preserves existing metadata entries.
- **Admin client:** It is now possible to edit metadata entries on versions.
- **Admin client:** Now has functionality for creating/updating indexes with the version info option, allowing the creation of versioned documents with this structure.
- **Command line client:** Extended the `check-federation`, `check-database`, `check-node`, and `check-library` command-line tools to support consistency checking of federation backups.
- Added support to keep duplicated transaction log files, per server node, in multiple locations.
- Added interface `XhiveLogConfigurationIf` to manage multiple transaction log files locations.
- Modified file extension (from `.log` to `.wal`) for transaction log files and `xhive_checkpoint.log` and `xhive_id.log` files as well.
- **Command line client:** Extended the `create-federation` and `add-node` command-line tools to support multiple log directories.

- **Command line client:** Added new commands (add-log-directory, remove-log-directory) to add and remove a secondary transaction log files directory.
- Fixed issue with multipath indexes that have a wildcard in namespace of subpaths (`//{*}el`) where updates to documents might not get propagated to the index.

Changes in release 10.4.1

- **Spring:** Added interface `com.xhive.spring.interfaces.XhiveDataSourceIf`. Customers can implement their own `XhiveDataSource` by implementing this interface. Minal sample implementation is shown in the Javadoc. Our implementation of this interface is class `com.xhive.spring.XhiveDataSource` which is made final
- No longer list reserved segments in xDB admin client as it cannot display information for non-existent data files.
- Fixed the problem that the bootstrap log record generated for `XhiveLibraryIf.removeBinding(String)` is wrong.
- Fixed a deadlock which occurs when a user thread and a system are both trying to kill the server.
- Fixed a bug which in rare cases can cause a federation backup to hang.
- **XQuery:** Fixed a multi-path index bug causing wrong score to be sent to remote clients in some cases.
- **XQuery:** Fixed a bug where usage of a for clause with 'allowing empty' could throw an `ArrayIndexOutOfBoundsException`.
- **XQuery:** Fixed 5 xquery pretty printer issues. Some of these issues could make an xquery unparsable.
- **XQuery:** Fixed namespace support in the `xhive:index-paths-values` XQuery option.
- **XQuery:** Fixed a bug where `fn:doc-available` on an empty library caused a NPE.
- **Command line client, Admin client, ant:** added support to access a federation through a federation set (description file or server). The path to such a federation contains a hash. The part before the hash should point to the set, the part after the hash should navigate to the federation. When creating a federation bootstrap file, its path should not contain a hash.
- Fixed a recovery issue which might happen if the system crashes at the moment of lucene segment shrinking.
- Fixed a recovery issue which might cause bootstrap file inconsistency if the system crashes.
- Fixed an issue with read-only federations that caused a NPE when obtaining segment information for detachable libraries.

Changes in release 10.4

- Kernel optimization. Implemented asynchronous model of bootstrap file (BF) updates which improved performance of database operations involving BF modifications (e.g. segment creation, set library to read-only, detach library, attach library) significantly.
- Upgraded to ANTLR 3.5, AspectJ 1.7.2, Fastutil 6.5.2, FOP 1.1, Guava 14.0, ICU4J 50.1.1, Jersey 1.17, Jetty 8.1.10.v20130312, JLine 2.10, Lucene 4.0.0, SLF4J 1.7.2, Spring 3.2.2.

- **Command line client:** Added global options `--stdout`, `--stdout-append`, `--stderr`, and `--stderr-append`, to allow redirection of standard output and standard error output to a file.
- **Command line client:** Added options to the backup command to specify included or excluded segments via a file, by using `--include-segments-file` or `--skip-segments-file` respectively.
- **Admin client:** Added new functionality: When executing an XPath/XUpdate/XQuery, if the results are idle, the underlying session will timeout. The user will then be asked whether or not to re-execute the action, and if not then the results tree will be disabled. The default timeout is set to 5 minutes and can be set through the options dialog.
- **Admin client:** You can now run/kill merge tasks on Multipath indexes using the context menu of the Index tab.
- Fixed a concurrency issue with the SegmentAccessInfo cache which may cause NPE while a front-end is retrieving the information for connection switch.
- Fixed several issues that could result in "incorrect magic number" exceptions.
- Fixed several concurrency issues with the segment cleaner.
- Fixed a data corruption issue related to not removing keys properly from deserialized extended full-text indexes. (Extended full-text indexes originate from X-Hive/DB 8 and older.)
- Added new functionality which allows consumers to subscribe to monitored statistics in regards to the cache buffer pool.
- Publishing MBean which displays cache buffer pool statistics.
- Added new method in XhiveLibraryChildIf interface for storing versioned documents, which enables historical searches on new versioned documents and libraries, using new xquery functions and new indexing options. To make library children searchable with the new `xhive:collection-*-date` xquery functions, they MUST be created with the new `makeVersionable()` method's 'queryable' parameter on true. Indexing is provided for by new options `XhiveIndexIf.VERSION_INFO` and `XhiveExternalIndexConfigurationIf.setStoreVersionInfo(true)`. The admin client currently only has functionality for creating/updating indexes with the version info option, it doesn't allow the creation of versioned documents with this structure yet.
- **XQuery:** XQuery 3.0 support.
- **XQuery:** Added `xhive:version-date-property` function to retrieve date properties from document versions. The function returns `xs:dateTime` values.
- **XQuery:** Fixed an xquery optimizer bug. An 'order by' query was optimized by an index join. The optimizer wrongfully assumed the result of the indexes already ordered.
- **XQuery:** Each xquery module of a specific namespace can now consist of multiple files. For this purpose, a new `resolveModuleImports` function is added to the `XQueryResolverIf` and the existing `resolveModuleImport` has become deprecated. When using abstract class `AbstractXQueryResolver`, the old `resolveModuleImport` is now called for each single import location. Depending on the existing customer implementation of function `resolveModuleImport`, this may result in different behavior of module resolving.
- **XQuery:** Fixed a multi-path index bug causing wrong score to be calculated in XQueries that used score but did not order by score.
- **Indexes:** Implemented support of Lucene IndexReader objects cache. It can significantly improve the performance of queries which use Lucene MultiPath index.
- **Indexes:** Fixed an issue with the `INCLUDE_DESCENDANTS` multi-path index option that did not distinguish among siblings with the same path even if `ENUMERATE_REPEATING_ELEMENTS` was set, causing incorrect content to be indexed.

- **Indexes:** Fixed the support for full-text logical queries ("LINE contains text 'assemble' f and 'draw'") in the multi-path index.
- **Indexes:** Fixed an issue with index adoption when creating multiple new indexes at once (via `XhiveIndexAdderIf`), which could lead to unusable indexes.
- **Indexes:** Fixed an issue where non-Administrator could not create/delete multi-path indexes.
- **Samples:** Added a Spring Web MVC sample.
- **Samples:** Made it clearer how to run the J2EE/Spring samples.

Changes in release 10.3

- Upgraded to Lucene 3.6.0.
- Switched from `java.util.logging` to `SLF4J`.
- Consistently referring to "Lucene index" and "External index" as "Multipath index" in API documentation.
- **Indexes:** Redesigned Lucene MultiPath Index (MPI) architecture. Now there are no external files stored outside of xdb segments, instead all lucene files are stored in xdb segments (as blob objects). The new architecture simplifies MultiPath Index transaction management and as a result indirectly fixed potential problems of complicated glueing of xDB and old MultiPath Index transaction models.
- **Indexes:** Support for fast adoption of B-tree indexes.
- **Indexes:** Support for super fast adoption of MultiPath indexes.
- **Indexes:** Full-text indexes can now also index empty elements.
- **Indexes:** Fixed concurrency issues in `XhiveIndexInConstructionIf`.
- **Indexes:** Fixed an issue with updating of value indexes on attribute change when a path value index is in scope.
- **Indexes:** Added support for index-level interval and CRON-style scheduling of final merges for multipath indexes.
- **Indexes:** Fixed a bug causing incorrect index updates when a path value index was in scope.
- **Indexes:** Do not index namespace declarations.
- **Indexes:** Fixed a potential NPE in value comparisons backed by multi-path indexes.
- **Indexes:** Advanced repair functionality for Lucene MultiPath Index.
- **XQuery:** fixed a concurrency issue in the parallel query implementation causing the thesaurus context library to be unusable.
- **XQuery:** added new function- `xhive:glob-documents()`. This enables us to use wildcards in order to retrieve documents. for example: `xhive:glob-documents(/Library*)` will return all documents under libraries which start with "Library"
- **XQuery:** added `xhive:full-path()` function to retrieve the full path of a library child in the database.
- **XQuery:** added support for executing XQueries with an undefined context item in `XhivePreparedQueryIf` and `XhiveXQueryQueryIf`.
- **XQuery:** added XQuery 3.0 features: group by clause, window clause, count clause, allowing empty (for clause), try/catch expressions, computed namespace constructors, private functions, switch expressions, named function references, dynamic function call, function tests, inline

function expressions, external variable declaration default values, annotations (no supported annotation implementations).

- **XQuery:** added XQuery 3.0 functions: `fn:function-name`, `fn:function-arity`, `fn:function-lookup`.
- **XQuery:** added collation support for order by clauses
- **XQuery:** the xquery optimizer now supports ordering by multipath indexes. When using multipath indexes, order by clauses support features ascending/descending, empty least/greatest and collations.
- **XQuery:** added parallel and non-parallel query evaluation support for queries with order by clauses addressing multiple roots or library sequences
- **XQuery:** `xhive:evaluate` supports specifying values for external query variables.
- **XQuery:** fixed multiple xquery optimizer bugs with negated conditions using `not()`.
- **XQuery:** fixed wildcard search on a filtered term caused nullpointer exception.
- **XQuery:** fixed xquery optimizer bug where the result of or-ing 2 ordered index results was considered ordered.
- **XQuery:** fixed a number of concurrency issues in the parallel query implementation.
- **XQuery:** The call function of `XhiveXQueryExtensionFunctionIf` used as highlighter has one additional argument containing position information. Depending of the custom implementation of the highlighter, this change may cause backward compatibility issues.
- **XQuery:** Fixed an XQuery optimizer bug that caused the `xhive:ignore-indexes` option to be ignored in certain cases.
- **Samples:** New sample `DeleteDatabase.java` shows how to delete database.
- **Samples:** New sample `MultithreadedOperations.java` shows how to perform multithreaded read/write operations and how to handle exceptions thrown in different threads, specifically how to handle `LockNotGrantedExceptions`.
- **Samples:** New sample `CreateMultinodeDatabase.java` shows how to create and handle multi-node databases.
- **Samples:** Replaced occurrences of `'xhive:fts'` and `'ftcontains'` with `'contains text'` within XQueries.
- **Samples:** Implemented correct session handling in samples, ensuring that we commit transactions at the end of samples unless there is an exception, if so we roll back to transactions.
- **Samples:** Removed `parser.setParameter("namespace-declarations", Boolean.FALSE)`; since we encourage users to use a namespace when parsing a document.
- **Samples:** Replaced old document parsing API with new DOM API. When parsing documents, use `LSParser.parseURI` instead of `XhiveLibraryIf.parseDocument`.
- **Samples:** When using an `LSParser` set an error handler to display errors which occur during parsing.
- **Samples:** Replacing occurrences of `GetAttribute` with `GetAttributeNS` and of `SetAttribute` with `SetAttributeNS`.
- **Samples:** Replaced old syntax for use of wildcards in XQueries with new one.
- **Samples:** Replaced old syntax for use of `'And'`, `'Or'` ... in XQueries with new one.
- Fixed an issue where the segment may remain in the segment cleaner's queue after being marked as unusable.
- Fixed an issue where the segment may remain in the segment cleaner's queue after being removed with `forceDelete()`.

- Added retry logic for RPC requests sent to libraries that are bound to more than one node. Now it will automatically attempt to send the request to next binding node if the previous one fails.
- Fixed a potential deadlock in XhiveLibraryIf.changeBinding(String).
- Fixed a repair tool bug where it may modify read-only indexes.
- Added a new attribute named 'reserved' for segments. Reserved segments are those whose data files are not present, however their segment records are kept in the bootstrap file so that those records are present when users restore a library which occupies them. Currently, only MultiPath Index segments can be 'reserved'.
- Replaced the JDK writeUTF()/readUTF() calls in socket communication with an implementation that can send/receive more than 64K bytes.
- Writes on temporary segments are allowed even if all writes are suspended.
- Added support for (optional) FIPS 140-2 encryption.
- The "format-pretty-print" LS Serializer option honors xml:space="preserve".
- **Admin client:** Fixed: option "Open in browser" could cause Null pointer exception.
- **Admin client:** Changed "Select active federation" shortcut to 'Ctrl+o'.
- **Admin client:** New look and feel, including new icons.
- **Admin client:** Added progress bar functionality.
- **Admin client:** Added cleaning lucene segment functionality. We will be able to clean segment from a database level or a library level
- **Admin client:** Index, Metadata, Properties and Multipath index subpath tables can be sorted.
- **Admin client:** Consistency check on blob no longer offers check index or check dom node options.
- **Admin client:** Removed "Cancel" button from error messages.
- **Admin client:** Adding more toolbar commands.
- **Admin client:** Changed administrator user icon.
- **Admin client:** Errors in database backup dialog no longer cause backup file corruption.
- **Admin client:** Added check box to "Create segment" dialog in order to apply unlimited storage.
- **Admin client:** Added check box to "Superuser login" dialog in order to remember previously entered password.
- **Admin client:** Connecting to a chained remote client no longer causes a freeze.
- **Admin client:** When connecting locally to a multi node configuration we no longer have access to server node management.
- **Admin client:** When creating a segment only active nodes are available for binding.
- **Admin client:** Added configurable path mapping functionality to restoration.
- **Command line client:** Added 'webserver-disable' option to 'run-server' command, which when used stops the webserver from being deployed on server startup.
- **Command line client:** When running a non primary node and providing a bootstrap file, that file will be used and not the one in the properties file.
- **Command line client:** Statistics commands can now be run while server is running in regular mode
- **Command line client:** When running consistency check commands, if no check options are specified then all options will be checked.

- **Command line client:** When running consistency check commands with options which are not supported, we no longer fail the check, but print out which options are not supported.
- **Command line client:** Added statistics-ls command, allowing printout of information on requested index and all of its subindexes.
- **Command line client:** Added repair-shrinksegments command, This allows us to to shrink index segments which are not referenced by the record but are still not empty.
- **Command line client:** Added repair-set-usable command. This allows you to set if a multipath index is usable. Meaning if the index will be used in a query or not.
- **Command line client:** When trying to create a segment and failing we will print out that segment was not created.
- **Command line client:** Added clean-library command. This allows you to clean lucene segments on a library level.
- **Command line client:** Added clean-database command. This allows you to clean lucene segments on a database level.
- **Command line client:** You can now return the previous lines in command either by backspace or the back key.
- **Command line client:** Each time you run the run-server command, it re-reads the xdb.properties file. Any changes made to xdb.properties will apply to the new server.
- **Command line client:** When running the command line client in windows we only cache some properties in as system variables.
- **Command line client:** Added tab completion for xDB commands.
- **Command line client:** Added relative and configurable path mapping functionality to restoration.
- **Command line client:** In order to enjoy full functionality when running Windows 2008R1 or Windows 7, you must install Microsoft Visual C++ Redistributed packages. Refer to the Installing section of the manual for specifics.
- Implemented XhiveExternalIndexMBean which is deployed when a primary server is deployed. This MBean supplies its consumer with the notifications on External Index merge operations, such as: final/not final, which index is being merged, state of the merge.
- **Consistency Checker:** Extended ConsistencyCheckerResult API: boolean isCheckApplicable(DatabaseCheckOptions option), boolean areChecksApplicable()
- **Consistency Checker:** When performing a check for CHECK_SEGMENTS_ADMINISTRATION_STRUCTURES options, now checks that size tree extents are contained within block tree.
- **Consistency Checker:** When performing a check for CHECK_SEGMENTS_ADMINISTRATION_STRUCTURES options, if there are extra files in a segment you will no longer receive a false positive.
- Federation backup with BACKUP_KEEP_LOG_FILES option is only applicable to standalone backup.
- New documentation: xDB Administration Guide (xdb_admin_guide.pdf). This Guide is intended as a convenience for readers who do not need specifics on java software development with xDB, for example system administrators and end users of xDB-based applications. Omitting developer-specific information, this Guide has about half as many pages as the xDB manual.
- Fixed a force-recovery issue where it may be unable to identify the correct affected data file(s) if the redo of a LOGTYPE_NAMEBASE_SPLIT record fails.
- Fixed a concurrency issue with the suspension of disk writes that caused XQuery queries to block because of waiting Lucene asynchronous tasks.

- Fixed Final Merge issue where log truncation was prevented, causing the disk with the logs to fill up.
- Increased installation defaults for minimum and maximum memory: 128MB and 256MB.
- Changed request type from byte to short to accommodate more RPC commands.

Changes in release 10.2.4

- Fixed an issue where the segment may remain in the segment cleaner's queue after being marked as unusable.
- Fixed an issue where the segment may remain in the segment cleaner's queue after being removed with forceDelete().
- Fixed a force-recovery issue where it may be unable to identify the correct affected data file(s) if the redo of a LOGTYPE_NAMEBASE_SPLIT record fails.
- Indexes: Fixed a bug causing incorrect index updates when a path value index was in scope.
- Indexes: Do not index namespace declarations.
- Indexes: Fixed a potential NPE in value comparisons backed by multi-path indexes.
- **XQuery**: fixed another xquery optimizer bug with negated conditions using not().

Changes in release 10.2.3

- Optimized the algorithm for collecting the detachable segments information during database startup.
- Fixed detachable libraries cache problem which could lead to lost updates.

Changes in release 10.2.2

- Revised the cache implementation for segment lists in such a way that the total overhead is reduced to minimum and will no longer grow linearly with the number of detachable libraries in the database.
- Fixed an issue where the segment may remain in the segment cleaner's queue on the original node after changeBinding() returns.
- **XQuery**: fixed xquery optimizer bug with negated conditions using not().

Changes in release 10.2.1

- Fixed a data corruption issue caused by segment cleaning operations not done in a serialized manner. Now each segment has its own separate cleaning session.

- No longer begin transactions on all secondary nodes at the start of a read-only transaction.
- Fixed a false alarm of `SEGMENT_BEING_DETACHED` in multi-node environment.
- Fixed a restore issue where existing data files of the segments that are excluded from the backup get overwritten by the empty files created by the restore procedure.
- Ensured that temporary pages are written in a temporary segment whenever possible, to avoid issues.
- Fixed restore issue which could lead to an unsupported operation exception.
- Fixed bug in creation of sorted indexes which could corrupt the database.
- Fixed a force-recovery issue where log records are mistakenly generated when segments are marked as unusable.
- Fixed NPE associated with checking the consistency of a server node.
- Fixed xDB log record issue which added zeroes to the end of the log and could, in case of a crash at that time, stop xDB server from starting up.
- Fixed bug in xDB installer concerning server and webserver listen addresses.
- Fixed bug which prevented some value index updates if a path value index was also in scope.
- Removed two public APIs: `XhiveFederationIf.registerReplicator(String, String)` and `XhiveFederationIf.unregisterReplicator(String, String)` which were never usable.
- Added support for custom loggers in `xdb.properties`.

Changes in release 10.2

- Upgraded to Lucene 3.4.0.
- Fixed a `LUCENE_SNAP_SHOT_TOO_OLD_ERROR` in Multi-path index which might cause query fail.
- Fixed an issue in Multi-path index which might cause search result inconsistent.
- Multi-path indexes now support namespaces and attributes in the sub-path specifications.
- Names in a namespace can be specified also using the Clark `{namespace}localpart` format in element name indexes.
- Added the `run-server-repair` command for running the xDB server in repair mode.
- Added Ant targets for generating OSGi-compatible xDB jars to `build.xml`.
- Fixed a concurrency issue in the algorithm for determining the database log lower bound for the active transaction set, which was causing fragmentation of concurrent indexes under certain conditions.
- Fixed a NPE in `xhive.evaluate()` when the context item was undefined.
- **XQuery:** Fixed regression where xqueries with a `[..[last()]]` predicate caused a nullpointer exception.
- **XQuery:** Fixed regression where xqueries with a conditional expression that contains a context independent subexpression caused a nullpointer exception.
- Added support for IPv6-style bootstrap URLs.
- Added support for XLink 1.1 Recommendation
- Added metadata support by Path indexes

- Added support for separately setting the web administration address (webserver-address).
- Fixed a NPE in `xhive:evaluate()` when the context item was undefined.
- No longer automatically rollback the transaction if the connection breaks during an RPC call. Instead, the transaction will remain open and the user will have to decide whether to abort it or not when receiving an `IO_ERROR` exception in such case.
- Fixed a potential deadlock (involving the control session and error channels) which might cause the remote server to hang in extremely rare cases.
- **XQuery:** added support for the XQuery Full Text weight option (works with multi-path indexes only).
- **XQuery:** added thesaurus support for full text xqueries.
- **XQuery:** added `XhiveXQueryFilterIf` profile information to Query Plan Profile.
- **XQuery:** `xhive:metadata` function can now be called on any context node. If the context node is a descendant of a document node, the function is executed on the metadata of the document node.
- **XQuery:** Path value indexes with metadata conditions are used by the xquery optimizer.
- **XQuery:** Fixed regression where xqueries with a `[..[last()]]` predicate caused a nullpointer exception.
- **XQuery:** Fixed regression where xqueries with a conditional expression that contains a context independent subexpression caused a nullpointer exception.
- **XQuery:** Fixed multiple cardinality matching bugs.
- **XQuery:** A `for`, `let`, `where`, or `order by` clause containing an updating expression now throws an exception.

Changes in release 10.1.2

- Fixed an `ArrayIndexOutOfBoundsException`; ICU collators are not thread save
- Replaced the `JDK writeUTF()/readUTF()` calls in socket communication with an implementation that can send/receive more than 64K bytes.

Changes in release 10.1.1

- Fixed a data corruption issue caused by segment cleaning operations not done in a serialized manner. Now each segment has its own separate cleaning session.
- Fixed a concurrency issue in the algorithm for determining the database log lower bound for the active transaction set, which was causing fragmentation of concurrent indexes under certain conditions.
- Support for IPv6-style bootstrap URLs added.
- Support for separately setting the web administration address added (webserver-address).
- Fixed a NPE in `xhive:evaluate()` when the context item was undefined.
- **XQuery:** Fixed regression where xqueries with a `[..[last()]]` predicate caused a nullpointer exception.

- **XQuery:** Fixed regression where xqueries with a conditional expression that contains a context independent subexpression caused a nullpointer exception.
- **XQuery:** fixed wildcard search on a filtered term caused nullpointer exception.
- No longer begin transactions on all secondary nodes at the start of a read-only transaction.
- Fixed a false alarm of SEGMENT_BEING_DETACHED in multi-node environment.
- Fixed a restore issue where existing data files of the segments that are excluded from the backup get overwritten by the empty files created by the restore procedure.
- Fixed a force-recovery issue where log records are mistakenly generated when segments are marked as unusable.
- Removed two public APIs: XhiveFederationIf.registerReplicator(String, String) and XhiveFederationIf.unregisterReplicator(String, String) which were never usable.

Changes in release 10.1

- **Administration:** a web based xDB Administrator Client has been introduced. It supplements the existing administration tools by also allowing various tasks to be performed from within a web browser. This new web based client is started on port 1280 by default when running the xDB server.
- Added new XhiveIndexInConstructionIf that allows creating a multi-path index without locking the library.
- **Indexes:** Added XhiveIndexListIf.definitionsToXml() and XhiveIndexListIf.addFromXml(XhiveDocumentIf) to export all index definitions from an index list, and to create indexes based on such files. Admin client, and Ant task support (see <batchindexadder/> and <listindexes/>) also added.
- **Ant:** new task <metadata/>, <librarydelete/> now accepts a "path" attribute for greater convenience, <listindexes/> and <batchindexadder/> were updated (see note about XhiveIndexListIf.definitionsToXml()).
- A XhiveSessionIf can now detect a Federation replacement. This may affect existing application test suites that replace federations and reuse the sessions.
- Increased stability of multi path indexes, which had several small issues fixed.
- **Indexes:** Significant library children traversal speed up (when using a library id or library name index). Libraries created before 10.1 need to rebuild their library indexes to take advantage of this.
- Increased speed of iterating over a XhiveNodeListIf
- **XQuery:** simple value indexes will now be used for path expressions without comparisons.
- **XQuery:** index supported order by expressions will now be correctly optimized when used multiple times. Also available in the Admin client.
- **XQuery:** new utility com.xhive.query.interfaces.XQueryPrettyPrinter to indent and highlight XQuery statements.
- **XQuery:** update queries on versioned documents now throw exception.
- **XQuery:** added extension function xhive:document-id to retrieve the id of a document.
- When restoring backups that contain multi-path indexes, if a PathMapper is used, it will also be used to remap the location of the multi-path indexes.

- **Known issue:** When indexing sub-paths (XhiveSubPathIf) with type DATE_TIME, timezones will be ignored in range searches. So for best results we suggest normalizing the date-time data to a single time zone.
- **Known issue:** Both XhiveAttrIf and XhiveElementIf do not currently implement ItemPSVI.getSchemaValue()
- Scoring in multi-path indexes is optimized for very large indexes and therefore we save memory by not storing text length for score normalization. While we (currently) don't offer any option to turn this on, you can effectively turn it back on by setting the score-boost of all sub-paths to the same value (the value must be different than the default). So by setting the boost of all sub-paths to 1.01 you can have better normalization in all your scores.
- **Tools:** many small fixes to XhiveFtpServer for commands such as store, rmdir, rename and list.
- **Tools:**the xdb command line: "xdb consistency-check" has a new option "deadobjects"
- **Tools:**the xdb command line: "xdb backup" has a new option "include-segments"
- **Documentation:**improved xDB manual structure, and minor content changes in the introductory chapters
- Fixed multi-path index recovery issue which might result in DEAD_OBJECT exception
- **XQuery:** parallel query evaluation support for queries addressing multiple roots
- Added support for new weighted score boosting model through XhiveWeightedFreshnessBoostIf interface
- Added support for full-text fuzzy search option
- Fixed segment cleaner thread starvation problem in concurrent environment
- Fixed SERVER_TERMINATED error caused by segment cleaner thread writing to READ_ONLY segments
- Fixed a data file handling issue which could cause server to hang while being shut down if some data files are missing
- Fixed NPE associated with layered import of modules.
- Increased stability of running xDB in multiple nodes
- Increased stability of xDB's client server protocol
- Upgraded the included Xerces-J library to version 2.11.0. Changes in this version affect the XMLResourceResolver: the value of XMLResourceResolver.getLiteralSystemId() is now equal to XMLResourceResolver.getExpandedSystemId().
- Hot backups taken on xDB 10.0 (or less) are not compatible with xDB 10.1. If you want to restore database/federation from such a backup you should do that using xDB 10.0 then cleanly shutdown server and upgrade to xDB 10.1. Another option is to create backups again on xDB 10.1.

Changes in release 10.0.2

- Fixed a concurrency issue in the algorithm for determining the database log lower bound for the active transaction set, which was causing fragmentation of concurrent indexes under certain conditions.
- Support for IPv6-style bootstrap URLs added.

- **Tools:** Ant tasks for creating new indexes (including `<batchindexadder/>`) have a new attribute "exists" to specify what to do in case of name collision. Notice that `batchindexadder` used to fail by default if a name collision happened, and now it will "skip" by default.
- **XQuery:** update queries on versioned documents now throw exception.
- **XQuery:** Fixed bug fulltext xquery with scoring can throw exception when using index.
- **XQuery:** Fixed regression where full-text queries against path-value indexes would search for tokens as a prefix and not as an exact match.
- **XQuery: Known Issue:** Path value indexes using multiple keys where one key is full-text, such as "a[b + c<FULL_TEXT::>]", will not use the index to resolve queries if the query does not specify the value of "b". Example "a[b and c contains text 'foo']". This is a regression from xDB 9.X. In xDB 10.0.1 this query would trigger a `NullPointerException`.
- Fixed NPE associated with layered import of modules.
- Fixed a performance issue in changing the state of or backing up a library with huge amount of documents beneath it.
- Fixed a bug where a secondary node might attempt to connect to itself when it should connect to the primary to report corrupt segments.

Changes in release 10.0.1

- **XQuery:** fix an issue with incorrect query results when intersecting index lookup results where individual nodes are subject to different namespaces.
- Avoid startup time `NODE_NAME_UNKNOWN` exception caused by a previous incomplete upgrade
- **Ant:** fixed Ant task `<multipathindex />` to honor the documented default values for options "lowercase" and "stopwords".
- Fix: the storage location of multipath indexes files (which are stored outside of the database) can now also be re-mapped through `PathMapper`.
- Fix several blacklist management issues which might result in `DEAD_OBJECT` or `ArrayIndexOutOfBoundsException`.
- Fixed `BufferOverflowException` associated with multi-path index backup.
- Fixed 2 issues associated with cross index merge 1. when adding subpath configurations after creating the multi-path index. 2. if the source index is an optimized index (does not contain non-final index).
- Fixed a memory leak associated with `XhiveNodeIf.getChildNodes()`.
- Fixed the '@since' version of `XhiveFederationIf.shutdown(String, String[])` in javadoc.
- Documented the behavior of `XhiveFederationFactoryIf.restoreLibrary(ReadableByteChannel, Collection<String>, String, PathMapper, XhiveRestoreCallbackIf)` when the second parameter is null.
- Read-only segments will now be temporarily made writable (if allowed by the file system) in case that the data on them needs to be upgraded from xDB 9 to xDB 10 during startup.
- Fix several sub-connection management issues which might result in `SWITCH_FROM_NON_PRIMARY_CONNECTION` and/or NPE.
- Fixed a memory leak associated with parallel query execution.

- Fixed NPE associated with query subresults merging.
- Fixed NPE associated with opening read-only federation.
- **XQuery:** fix an issue with incorrect query results when intersecting index lookup results where individual nodes are subject to different namespaces.
- **Tools:** fixed an issue with the xDB installer and launcher script on Solaris.
- **Tools:** documented usage of xDB together with Maven 2.
- **Spring:** integrated the xDB Spring support into the main distribution. See the manual section "xDB and Spring" for details.
- **Administration:** documented and unified use of logging within xDB.
- **XQuery:** various settings from query now also apply to queries run in **xhive:evaluate()**

Changes in release 10.0.0

- An xDB federation can now be distributed over multiple server machines. See manual chapter "Configuring Multiple Backend Servers".
- There is an interface to find information about an existing backup file. See `XhiveFederationFactoryIf.getBackupInfo`.
- Added possibility of compressing Text and CDATASection of imported nodes. See `XhiveNodeCallbackIf` for details.
- Add new method `XhiveLSParserIf.parseIntoDocument` to parse into an existing document. This method does not have all the features of `LSParser.parseWithContext`, but is more efficient in time and memory use.
- Databases can now be created with a concurrent root library. See `XhiveFederationIf.DatabaseOption.CONCURRENT_ROOT`.
- An xDB instance can now be enlisted as a resource with an XA transaction manager. See `XhiveDriverIf.getXAResource`.
- All new namespaces are now concurrent. The options `XhiveLibraryIf.CONCURRENT_NAMESPACE` and `XhiveLibraryIf.DOCUMENTS_DO_NOT_LOCK_WITH_PARENT` are now ignored and have been deprecated.
- Added RPC tracing support when running in client/server mode. See `XhiveSessionIf.enableRPCTracing`.
- For debugging purposes, a driver can now be given a name using `XhiveDriverIf.setName`.
- Restore operations can now optionally overwrite existing files. See `XhiveRestoreCallbackIf`.
- Changed hash code definition for `XhiveXQueryValueIf`. Atomic number values will now map to the same hash code even if they are not of the same type (xs:double, xs:decimal, xs:integer etc.). This means that they will be considered equal in hashmaps and similar containers.
- Index lookups will now be done with values in sorted order to enhance cache use. Predicate evaluation with equality comparisons on very large sets is now faster.
- Added a cache of search string tokens. It avoids retokenizing the same string if it is used twice within a single query.

- For performance reasons, all index lookups from XQuery will be performed ascending unless an order by expression explicitly sorts the result in the other order. This may cause different result orders for queries using "unordered {}".
- New Ant tasks: <metadata/>, <checkdatabase/>, <checkfederation/>, <checknode/>, <checklibrary/>, <multipathindex/>, and <xquery/>.
- Added new check-federation, check-database, check-node and check-library command-line tools.
- The admin client now shows a paged view for libraries with many children. This avoids OutOfMemoryErrors in the client.
- Added database/federation consistency checker interfaces XhiveConsistencyCheckerIf/XhiveFederationConsistencyCheckerIf. They allows you to check administrative and data pages consistency.
- Added support for consistency checker in admin client.
- Integrated J2ee module into xDB. Added samples how to use module from EJB and spring frameworks.
- Added support for times filter of XQFT spec. available at <http://www.w3.org/TR/xpath-full-text-10/#doc-xquery-FTTimes>
- Added support for 'at start/at end/entire content' positional filters of XQFT spec. available at <http://www.w3.org/TR/xpath-full-text-10/#ftcontent>
- Implemented XQFT feature parity for all full-text indexes.
- If a full-text index analyzer returns multiple tokens with the same position, the tokens will be joined in an OR query.
- Marked XhiveCCIndexIf as deprecated to clearly discourage its usage.
- New index type: multi-path indexes. See the manual for more information.
- **XQuery:** Added an API XhiveScoreBoostFactorIf to boost library child score.
- **XQuery:** fixed an ArrayIndexOutOfBoundsException with order by expressions and multiple for clauses
- **XQuery:** added optimization for conditions that check multiple children using starts-with in a some ... satisfies loop, e.g. for \$x in ...
where some \$y in \$x/author/last satisfies starts-with(\$y, 'Smi')
return \$x
Such conditions can now use index scans.
- **XQuery:** extended support for libraries. XQuerys can now use libraries in sequences ("(doc('a'), doc('b'))//foo") or variables bound by declare variable or let statements, and path expressions will still pick up indexes at the library level.
- **XQuery:** order by expressions are now supported across multiple libraries or documents. E.g.
for \$x in (doc('a'), doc('b'))//test[@price]
order by \$x/@price
return \$x
will run as an index supported order by if there is an index on @price on 'a' and 'b'.
- **XQuery:** Access to variables using positional predicates (e.g. \$variable[5]) is now handled more efficiently.
- **XQuery:** Added an API to get the query plan for an XQuery, either as a static description or including profiling information after execution. See JavaDocs for XhiveXQueryQueryIf.getQueryPlan() and XhiveXQueryResultIf.getQueryPlan().
- **XQuery:** relaxed the Update Statement Placement rules by default, enable strict checking through XhiveXQueryCompilerIf.setStrictUpdateExpressions.

- **XQuery:** replace value of \$node with \$value will now always convert \$value to a single string instead of appending nodes - this reflects a change in the specification. Also applies to xhive:replace-value-of WARNING: this might break existing XQuery code!
- **XQuery:** Java module functions can now also declare parameters as XhiveElementIf, XhiveAttrIf or XhiveDocumentIf, booleans, and java.util.Calendar.
- The toString() and toXml() methods of DOM nodes will now serialize the XML fragment with namespace support, i.e. missing declarations will be inserted where necessary.
- **XQuery:** errors triggered using the function fn:error(\$name as xs:QName?, \$message as xs:string, \$values as item(*)*) will now cause the exception com.xhive.error.xquery.XhiveXQueryUserException. This exception class provides accessors for the QName and values list.
- **XQuery:** fixed a large number of issues with the regular expression engine. Changed exception subtype for illegal replacement strings from XhiveXQueryParseException to XhiveXQueryErrorException.
- Moved configuration file xdb.properties to the subdirectory conf, adjusted it to be a proper Java properties file. This will be done automatically by the Windows installer.
- Server log output will be written to log/server-out.log and log/server-err.log instead of the bin directory.
- New implementation of parallel queries using lazy evaluation strategy.
- New samples: CreateMultiPathIndex, LibrariesAsVariable and BoostLibraryScore.
- **Fix:** updates of indexes indexing attributes of nodes without children could fail on some cases.
- **Fix:** problem with wildcard queries evaluation that could crash the query, or that could cause a given node id to be returned more than once from the index.
- **Fix:** memory leak when using XQFT's window and distance operators.
- **Tools:** The xdb xquery command can now read the XQuery from a file, using the -file option. See xdb help xquery for details.
- **XQuery:** adjusted the function signature of fn:put() to always require the second argument \$uri. Use the empty string to store documents without a name.
- Known issue: Multi path indexes are implemented as external indexes, and the backup restore operation of these indexes do not make use of PathMapper.

Changes in release 9.0.10

- Introduce a new exception type 'SEGMENT_BEING_DETACHED' which indicates that a read-only transaction is attempting to access a segment which was detached by a yet-to-commit transaction when it began.
- Fix a problem with deadlock detection algorithm which may unnecessarily abort transactions when some locks are already downgraded or released.
- Fix a problem where new read-only transactions could still be added to the transaction table while suspended.
- Fix a problem where potential deadlock may occur when a detaching transaction is trying to suspend new read-only transactions.

Changes in release 9.0.9

- **XQuery:** fix an issue with incorrect query results when intersecting index lookup results where individual nodes are subject to different namespaces.
- **Tools:** fix an issue with the xdb and setup.sh scripts on Solaris.

Changes in release 9.0.8

- **Fix:** do not ask LogManager accessing pages in read only transactions.

Changes in release 9.0.7

- **XQuery:** fix a NullPointerException with document filters that (correctly according to Javadoc) set individual items to null in the filtered sequence.

Changes in release 9.0.6

- Fix recovery problem when attempting redo of a change of segment properties on a segment that was already deleted.
- Added supportprefixwildcard option to <fulltextmetadata/> Ant task.

Changes in release 9.0.5

- Fix problem that caused obsolete log files on replicators to be retained.
- Empty elements and metadata fields are now correctly flagged as errors when an attempt is made to add them to a numeric index.
- **XQuery:** fixed a NullPointerException in the built-in XQuery function fn:error(\$qname, \$message, \$items) triggered when the \$items parameter was present but the empty sequence.
- **XQuery:** fixed a ClassCastException triggered when joining results from an index optimized order by lookup over multiple keys with an index lookup over a single key.

Changes in release 9.0.4

- Fix versioning of entity references with unresolved values, and of different entity references with the identical value.

Changes in release 9.0.3

- **XQuery:** Fix a stack overflow triggered by index lookups with conditions like "(a or b) and (c or d)" where both sides of the conjunction could be index optimized.
- **XQuery:** fixed a NullPointerException when using an XhiveXQueryResolverIf and calling collection() without parameters

Changes in release 9.0.2

- Fix rare data corruption problem that could occur when a document with entries in a concurrent index was removed in two threads simultaneously.
- **XQuery:** Added support to convert values of type java.util.Date, java.util.Calendar, java.sql.(Date|Timestamp|Time) into XQuery values when used in variables or external functions.
- **XQuery:** added asQName() and asCalendar() methods on XhiveXQueryValueIf
- Command line: fixed the unintended behaviour of "xdb import" to always create the complete path of an imported file relative to the current working directory in the database. Libraries will now only be created when importing directories, and only for paths relative to the imported directory.
- **XQuery:** added com.xhive.query.interfaces.XQueryResolverIf. Set a resolver on an XhiveXQueryCompilerIf or on an XhiveXQueryQueryIf to control module, schema, and document resolution. samples.manual.XQueryResolver shows how to use the interface.

Changes in release 9.0.1

- If you are upgrading an existing federation from version 9.0.0 (only) please read repair_searchable.txt first.
- Fixed Path index updates regression. Updating attribute values in path indexes would fail to actually update the index in some circumstances. Path indexes which had attribute components should be rebuilt to assure consistency.
- Fixed XA transactions recovery bug which happens during distributed transaction recovery managed by global transaction manager.
- The method XhivePreparedQueryIf#getExternalFunctionNames() will now also return the names of external functions from imported modules.
- Fixed XhivePreparedQueryIf#getExternalVariableNames() to properly return the external variable names.
- Fixed failonerror usage with createdatabase task. Fixed behavior of overwrite="newer" in parse/upload tasks. Also improved the options support for many Ant tasks, and fixed documentation issues with the adduser and addgroup tasks.
- **XQuery:** fixed two issues with positional variables in for clauses where positional variables would get a wrong value or cause an ArrayIndexOutOfBoundsException

- Command line: added the `cd` and `mv` commands to the command line client
- **XQuery**: fixed an issue where tail callable functions might produce values in the wrong order if the same call site would be evaluated in parallel (e.g. through mutual recursion).

Changes in release 9.0.0

- Samples have been added to show new functionality.
- Product has been renamed to EMC Documentum xDB, all references in API doc to xDB version 8.0 (and before) should be read as X-Hive/DB 8 (and before).
- The installation layout has significantly changed. It is best to perform a complete deinstallation and reinstallation. In particular, the Windows service is now called 'xdb-server', and all settings have been moved to 'xdb.properties' in the installation home directory. On Unix an installation path is no longer needed as it will install xDB in the directory the installation packages is unzipped.
- Non-concurrent indexes can now be created as compressed indexes. This can reduce disk usage, particularly for full text indexes.
- The W3C Element Traversal Specification has been implemented.
- Add new LSParser and LSSerializer option "xhive-insert-xmlbase", to add xml:base attributes on the top level elements of external parsed entities.
- Added new event types in XhiveDriverIf.XhiveDriverObserverIf to allow notification of loss and restore of connection to server.
- Added new method to XhiveXQueryPolicyIf to allow control over schema imports.
- Add new XQuery extension function xhive:highlight to allow userdefined highlighting function to get tokens searched for.
- Add new API XhiveFtsUtilIf.compilePattern to match patterns with full text search wildcards against terms.
- The included version of ICU4J has been upgraded to 4.0.
- The included version of Lucene has been upgraded to 2.4.0.
- The included version of XML Beans has been upgraded to 2.4.0.
- The included version of Apache FOP has been upgraded to 0.95.
- The included ant.jar has been upgraded to version 1.7.1.
- The Ant task <upload/> originally accepted an attribute parameter "xmlextentions", which misspells the word 'extensions'. This task has been modified to accept a "xmlextensions". The previous misspelled form is now unsupported.
- Adjusted the XQuery Update spec implementation to the current W3C Candidate Recommendation, available at: <http://www.w3.org/TR/2008/CR-xquery-update-10-20080801/>
- Implemented the rules for XQuery Update expression compatibility from <http://www.w3.org/TR/xqupdate/#id-upd-apply-updates> This disallows duplicate renames, replaces, and puts of elements/docs within one query. This might break some existing applications, in particular those using "move" operations by first deleting a node and then inserting it somewhere else, if the nodes are covered by a unique index. Please perform the move in two XQuery statements, or use the new functions xhive:move(\$target, \$sources) (insert into) and xhive:move(\$target, \$anchor, \$sources) (insert before \$anchor). Also see the manual section "XQuery Update Syntax".

- Implemented an interactive shell for the database. The shell can be started using the executable 'xhive' or 'xhive.bat' respectively. It replaces the older command line tools (like XHBackup), which are still in place for backwards compatibility. All previous commands are supported, plus new commands to manage database contents (list, remove, import, XQuery, ...). See 'xhive help' or the Administering xDB section in the manual for more details.
- XhiveXQueryCompilerIf has been extended to support setting default values for certain XQuery prolog settings, including external functions, imported modules, variables, options, etc. See the JavaDocs for more details.
- The default revalidation mode has been set to 'skip' (used to be lax) for performance reasons. The setting can be overridden within the query using 'declare revalidation (skip|lax|strict)' or using XhiveXQueryCompilerIf.setRevalidationMode(...).
- Fixed a bug where specifying the Unicode codepoint collation would rather use a Collation for the current system locale instead of the correct codepoint comparison method.
- Implemented XQuery Update copy/modify expressions
- Changed xhive:insert-document() and fn:put() to overwrite documents instead of giving an error
- Deprecated XhiveXQueryValueException, catch XhiveException instead. This exception used to signal the error that a full text conjunction setting had an erroneous value in 'xhive:fts-implicit-conjunction'.
- The XHServer tool now registers an instance of XhiveFederationMBean with the platform MBean server.
- Libraries (XhiveLibraryIf) can now be used as external values within XQuery, as external variables (via setVariable(...)), when returned from extension functions, Java modules, etc.
- Implemented compression of text and CDATA nodes. Added API which allows to specify which nodes to compress.
- Implemented recovery of XA transactions after database crash.
- Partially implemented XQuery Full Text Specification, available at: <http://www.w3.org/TR/xpath-full-text-10/> The list of supported features include: logical full-text operators, wildcard option, anyall options, positional filters and score variables.
- Support for scoring has been added to our Full Text search engine. The user may influence the score by changing the Similarity measure used to compile the score. See the manual for details.
- Added optimization of 'order by \$score' clause: when the result is coming from index pre-ordered by score then order by clause does not perform sort operation. The optimization is implemented for parallel execution of full-text query as well.
- Added new debug option, optimizer-debug, to help understand why the optimizer chose (or did not choose) a particular query plan. Added a new tab to the Admin client for the optimizer-debug output, as the output can be very verbose we don't want it to interfere with the output from any other debug option.
- Introduction of detachable libraries, detachable libraries can be detached and attached from the database. See the manual for details.
- Detachable libraries can be backed up and restored individually. See the manual for details.
- Fixed a potential security issue with Java module imports and XhiveXQuerySecurityPolicyIf. If the security policy was only set after parsing the query, Java access would not be prevented.
- Fixed a bug in the matching of Phrase queries using wildcards in one of the terms.
- Disallowed rebuild the "Library ID index" of concurrent libraries, as this would lead to losing track of the library children.

Known issues and limitations

The xDB manual usually discusses specific issues and limitations inside relevant chapters, topics or sections, including:

- Performance: [using multiple disks](#), page 77
- [Multipath index limitations](#), page 160
- XQuery [Full-text search limitations](#), page 195
- XQuery using java: [function name limitations](#), page 219
- XQuery [parallel query limitations](#), page 216
- [Multi-node considerations](#), page 302

The list below presents additional known issues and limitations that may affect your use of the product, and are not discussed elsewhere in the xDB manual.

- Restore: if the bootstrap contains Windows file paths like `C:\...\data\XHivedb-default-0.XhiveDatabase.DB`, federation restore of the bootstrap file goes wrong in *nix environments as the paths are not interpreted correctly (XDB-2626).
- The xDB installation process copies the manual files (PDF and HTML versions) from the distribution to a location where hyperlinks to java samples and javadoc should work after successful installation. If used from a different location, these links may not work (XDB-3315).
- The 'statistics-ls' command doesn't work on a multinode environment (XDB-3478).

Quick Start

This chapter contains the following topics:

- [Getting a quick start with xDB](#)

Getting a quick start with xDB

For developers who want to get up and running quickly with this version of xDB, this section briefly describes the minimal necessary steps: installation, creating your first database, and running a sample command to verify the installation. For more detailed information, see [Installing xDB, page 49](#), and the `readme.txt` file of the distribution.

Installing and running xDB requires the Sun JDK 7 or a fully compatible Java Virtual Machine (JVM).

1. Install xDB.

- On Windows, you must be logged on with Windows administrator access privileges. The Windows installer can upgrade an existing xDB version 9.0 or later (see [Upgrading xDB on Windows, page 57](#)). If you have a previous version of xDB running, either use different directories and port numbers for the new installation, or uninstall the previous version.

Run the `xdb_setup.exe` file and follow the instructions on the screen.

- On UNIX, you can install xDB under any account. If you have a previous version of xDB running, either use different directories and port numbers for the new installation, or uninstall the previous version.

Extract the distribution and run `sh setup.sh` and follow the instructions on the screen.

UNIX installation requires some post-installation steps (see [Installing xDB on a UNIX platform](#)).

2. The use the code samples that the xDB manual refers to, create a demo database as follows:

a. Start the Admin Client.

You can start it with the `xdb admin` command in the `bin` subdirectory of the xDB installation. On Windows, you can also launch it from the Windows start menu.

b. Select menu option **Database > Create database** to create a database.

c. Enter `united_nations` as database name, the superuser password as entered during xDB installation, and administrator password `northsea`. The other fields should be left unchanged.

d. After creating the database, you can close the Admin Client, unless you would like to use it later to view the results of running code samples, or to perform other actions.

3. Run a sample.

- a. Open a command prompt and navigate to the `bin` subdirectory of the xDB installation.

- b. Use the **xhive-ant** command to insert two documents into the database:

```
xhive-ant run-sample -Dname>manual.StoreDocuments
```

If the command runs successfully, a message appears stating the number of documents stored in the database.

Related links

[Pre-installation requirements](#)

[Installing xDB on a Windows platform](#)

[Installing xDB on a UNIX platform](#)

Introduction

This chapter contains the following topics:

- **xDB Overview**
- **General features**
- **Linking documents with XLink**
- **Versioning and branching**
- **Administration tools**
- **Logical architecture**
- **Internal structure: databases, segments, files and pages**
- **Detachable libraries**
- **Managing users and groups**

xDB Overview

xDB stores XML documents in an integrated, highly scalable, high-performance, object-oriented database. It exposes the database and its contents to software developers through an application programming interface (API). xDB is pure Java. Using xDB, software developers can build custom XML content management solutions that offer high-speed storage and manipulation of very large quantities of XML documents, and are fully tailored to the exact requirements of any given application.

Typically, software developers embed xDB JAR files within a calling application, which usually is a web application running within a Java application server. Through the xDB Application Programming Interfaces (APIs), developers can create front-end applications that concurrently call their backend xDB server to perform database sessions, retrieve XML documents, store XML documents, execute xQueries, etc.

General features

The page server

A backend server for an application is called a *page server*, because its purpose is to transfer data pages to front-end applications (also called *client applications*), which locally perform operations on the data.

In environments where all database accesses are done from a single application server, performance is usually best when the page server runs in the same JVM as the server application.

A backend server that combines being a page server with other tasks is called an *internal server*.

A page server that has no other purpose than being a server is called a *dedicated server*. A dedicated server, with client/server communication over TCP/IP, can offer better scalability than an internal server. In a simple internal server setup, one single web application can access the data on the page server directly, and if other web applications also need access to the data, the first web application can run an internal server for them. The larger the scale, the complexity and the number of different frontend web applications that share a single page server, the more likely it becomes that a dedicated server is preferable.

Clients and servers

The optimum numbers of clients and page servers will depend on the characteristics of the solution, including how data intensive it is. As the numbers of data retrievals or ingestions increase, more front-end application servers and/or more backend server nodes may become advisable. XQuery operations retrieve data pages from server to client side, and then process the query on the client side. So, query-intensive applications may benefit from additional servers both at client side and at server side. Ingestion operations parse XML documents on the client side and then pass newly created data pages to server side. So, in case of a high ingestion workload, client and server sides may both benefit from additional servers.

If multiple backend servers are required, there is a choice between two different features:

- *Replication* dynamically maintains one or more complete copies of an entire 'master' data set on one or more separate page servers. These copies are called replicas. Read-only transactions and online backups can be offloaded to the replicas to distribute query load. For more information, see [Replicating Federations, page 289](#).
- *Multi-node* distributes a data set over multiple *node servers*, using [Detachable libraries, page 46](#). It allows the application workload to be spread over multiple backend servers. For more information, see [Configuring Multiple Backend Servers, page 297](#).

Supported standards

Implemented and extended recommendations of the *World Wide Web Consortium (W3C)* for querying, retrieving, manipulating, and writing XML documents include:

- [Document Object Model \(DOM\)](#)

- Level 1
- Level 2 (Core and Traversal)
- Level 3 (Core and Load/Save)
- [The eXtensible Stylesheet Language - Transformation \(XSLT\)](#)
- [XQuery](#)
- [XPath](#)
- [XLink](#)
- [XPointer](#)

Implementation of an extended DOM level 3 interface provides for manipulating content, structure, and style of documents. All DOM level 3 functionality is supported, including functions for retrieval, modification and navigation within XML documents.

Since DOM level 3 does not support XML collections for handling more than one document, extended API functions provide support for processing multiple documents simultaneously. Documents are collected in libraries, which are implemented as DOM nodes. You can store libraries within other libraries in the same way that you store documents within libraries. All operations on documents (including XQuery queries) can also be performed on libraries.

A transformation engine that uses XSLT, a language for transforming XML documents into other XML documents, makes it possible to transform XML into such formats as HTML or WML. You can also publish to the PDF format.

XML Query Language (XQuery) has a string syntax that can address any type of information in an XML document. XQuery can make selections based on conditions and construct new structures based on queried information. The XQuery query engine implementation supports XPath and XPointer queries. For more information, see the chapter about [XQuery](#), page 173.

XLink is a W3C recommendation that enables links between XML documents. For more information, see [Linking documents with XLink](#), page 40.

Indexing

Various different types of indexes enable faster data access and increase the performance and scalability of applications. For more information, see the chapter about [Indexes](#), page 150.

Non-XML import and BLOB storage

Data can be imported and exported. The included SQL Loader uses Java Database Connectivity (JDBC) to import data from relational databases, sequential files and other non-XML sources. An integrated version of the Xerces parser imports XML documents.

In addition to XML documents, a database can store image files, sound files, Microsoft Office files, and other file formats as Binary Large Objects (BLOBs). Storing BLOBs and XML documents in the same database allows managing all resources for a specific project or product in one uniform way.

Session and transaction control

A transaction mechanism ensures that changes and updates to the database are completed harmoniously across the system. If a transaction conflicts with other transactions, all transactions that take place within one session can be committed or rolled back.

The transaction mechanism complies with the ACID database properties:

- **Atomicity:** either all actions in a transaction succeed and are made persistent in the database, or none of the actions succeed.
- **Consistency:** the view of a database within a transaction is coherent: all read actions on a particular part of the database return the same value.
- **Isolation:** changes that are made in one transaction are not visible in concurrent transactions until the transaction is committed.
- **Durability:** when data is modified and the transaction succeeds, the data is certain to be written to the disk.

Access control

User access control is provided in the form of authorization and security support for [managing users and groups, page 47](#).

Linking documents with XLink

XLink enables simple links between XML documents, equivalent to `<a href>` links in HTML, as well as more complex links, known as extended links. For example, extended links can be used to create one-to-many links, and to add semantic meaning to links. For more information about XLink, see the [W3C XML Linking Language \(XLink\)](#) documentation.

All XLink information is stored in attributes. The **xlink:show** attribute describes how the content is displayed, while the **xlink:type** attribute describes the type of the XLink. The **xlink:href** attribute defines the link target and can be defined using URIs and XPointer. For example,

```
xlink:href="/UN Charter/UN Charter - Chapter 4#xpointer(/chapter[1]/title[1])"
```

defines the first title of the first chapter of the UN Charter - Chapter 4 document in the UN Charter library as the link target.

Versioning and branching

Versioning

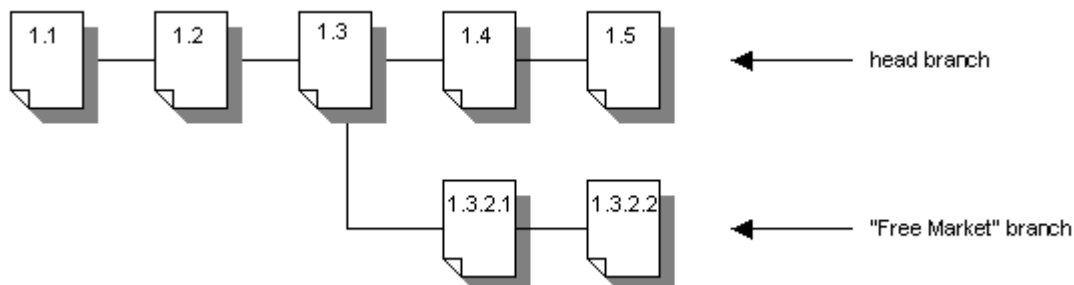
For applications that need to store multiple versions of documents and BLOBs, xDB offers linear versioning with branches. Storing multiple document versions allows users to track the changes in a document and restore older versions. The latest version of a versioned document is used for retrieval, traversal, querying and indexing. Earlier versions are only searchable if they were created with a 'queryable' option.

Branching

A version branch is a sequence of versions which has been separated from the main sequence of versions. Branches are typically used when multiple users are working in parallel on the same documents. If a document is checked in and the version has one or more successors, xDB automatically creates another version branch.

In the example shown below, two users check out document version 1.3 from the head branch. User A modifies the document, then checks in the document and xDB creates version 1.4. When user B checks in, the head branch already contains version 1.4 of the document and xDB automatically creates another branch under the 1.3 branch.

Figure 1 Branching documents



xDB automatically numbers versions and branches, as follows:

- Versions, for example 1.1, 1.2, 1.3, ...
- Branches, for example 1.x.2.1, 1.x.4.1, 1.x.6.1, ...
- Subbranches, for example 1.x.y.1: 1.x.y.1.2.1, 1.x.y.1.4.1, 1.x.y.1.6.1, ...

Administration tools

The xDB distribution includes a number of helpful tools, primarily intended for administrative use in a development environment.

The *Admin Client* (also known as *administration client* or administration tool) offers a database explorer that displays database structure and content, and provides access to a wide range of management functions, such as backup and restore, user authorization, and checking the physical and logical consistency of a database or federation. For more information, refer to [Admin Client, page 227](#).

Some key administrative database functions are provided by the new [web client, page 246](#).

Administrative tasks, including backup and restore, can be performed from the operating system command line, terminals or scripting languages using [Command-line tools, page 246](#).

Some administrative features are also accessible through xDB Ant tasks.

Logical architecture

A page server works with a federation, which contains one or more databases. Databases can hold data of various kinds, including XML documents, user accounts and other [database objects, page 43](#).

Federation and database

A *federation* is a container for related databases, to which it provides a single server connection. A federation is associated with a [superuser account, page 42](#), and a specific location for [transaction log files, page 43](#). Applications can connect to the federation's database driver either directly or remotely. When a calling application creates a driver, it specifies the required federation or page server by means of a [bootstrap, page 69](#).

If multiple applications need access to the same federation, one application will access the federation directly, and act as a server for the other applications.

A federation can contain as many databases as needed.

Federations and databases can be created and managed manually with the [Admin Client, page 227](#), or with the [Command-line client, page 246](#).

Creating a new database requires the password of the superuser account, the password of the administrator account and the name of the database you want to create.

Note: xDB database names, user names, and passwords are case sensitive. For example, Xhive and XHIVE are treated as different database names. Passwords must be alphanumeric and from 3 through 8 characters long.

Superuser

A federation has one superuser account with the user name **superuser**. The superuser account for the default federation is created during the installation process, and enables initial database configuration. The superuser can create and delete databases, and perform administrative operations such as setting the license key and performing backups.

The superuser cannot access regular data such as libraries and documents.

Transaction log files

Write Ahead Logging (WAL) is used to ensure that federation data can be recovered. Data is committed to the write ahead transaction log *before* it is written to the actual database files.

Transaction log files are numbered sequentially, and written to the federation's log directory, which was specified when the federation was created.

To protect against corruption it is possible to configure the system to keep multiple copies of transaction log files. For more information refer to [Duplicated transaction log files, page 275](#).

By default, transaction log files that are no longer needed for recovery are automatically deleted. To allow incremental backups, the `keep-log-files` option of the federation must be turned on, so that obsolete log files are only deleted after a backup.

[Creating and restoring backups](#)

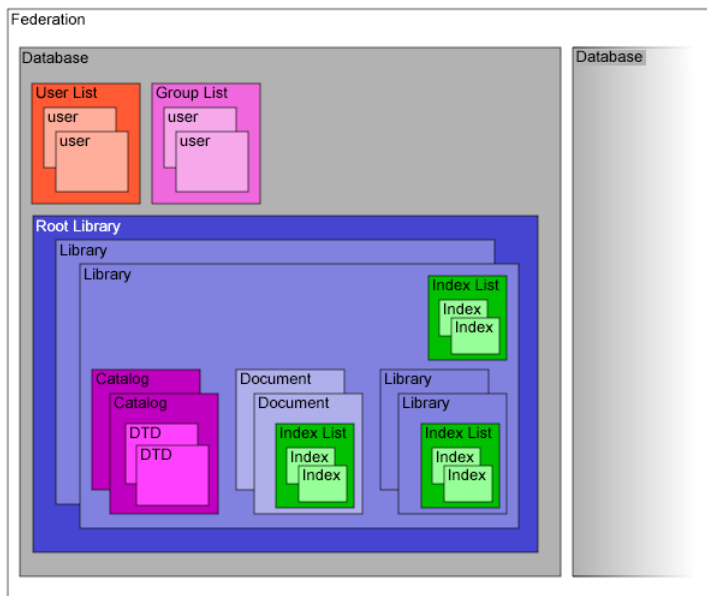
[Running incremental backups](#)

[Restoring lost data from log files](#)

Database objects

A federation can contain one or more databases. A database can contain various types of objects, including users, user groups, libraries, documents, indexes, catalogs and BLOBs.

Figure 2 Database objects



Users and user groups

When creating a database, the superuser must create a user account for the database administrator. The database administrator can create user accounts for all other database users. Every user has a database user account with a unique user name, with a password for access control. Each user account is assigned a set of permissions that specify the level of access the user has to the database.

A group is an object in the database that contains one or more users. Groups can be used to assign the same access rights and privileges to multiple users.

Each database has a user list and a group list.

Libraries

A library stores documents and other libraries in a hierarchical structure. Libraries can be stored within other libraries; the nested structure of libraries in a database is similar to the nested structure of directories or folders within a file system. The topmost library in the hierarchy is called the *Root library*. A database has exactly one root library, which is created automatically.

From a data modeling perspective, you can think of a library as a folder that can contain XML documents, indexes, catalogs and BLOBs, as well as sub-libraries. Libraries are implemented as DOM nodes, and all operations on documents, including XQuery queries, can also be performed on libraries.

Documents

A document is an object that stores XML data. The system can handle both valid (that is, conforming to a structure defined in a DTD or XML Schema) and well-formed XML documents.

Indexes and index lists

Indexes can be used to speed up queries. Available indexing methods include full-text, multipath, value, ID attribute and element name indexes.

An index list provides a list of all the indexes of a library or document.

Catalogs and validation

XML documents are validated against document type definitions (DTD) or XML Schemas, collectively called *models*. Documents using an XML schema are validated differently than documents using a DTD. The XML validation process can store DTD or XML schema information as an abstract schema model (ASModel) in a catalog, which is linked to a library. By default, only the root library has a catalog. Each model in a catalog has a unique ID.

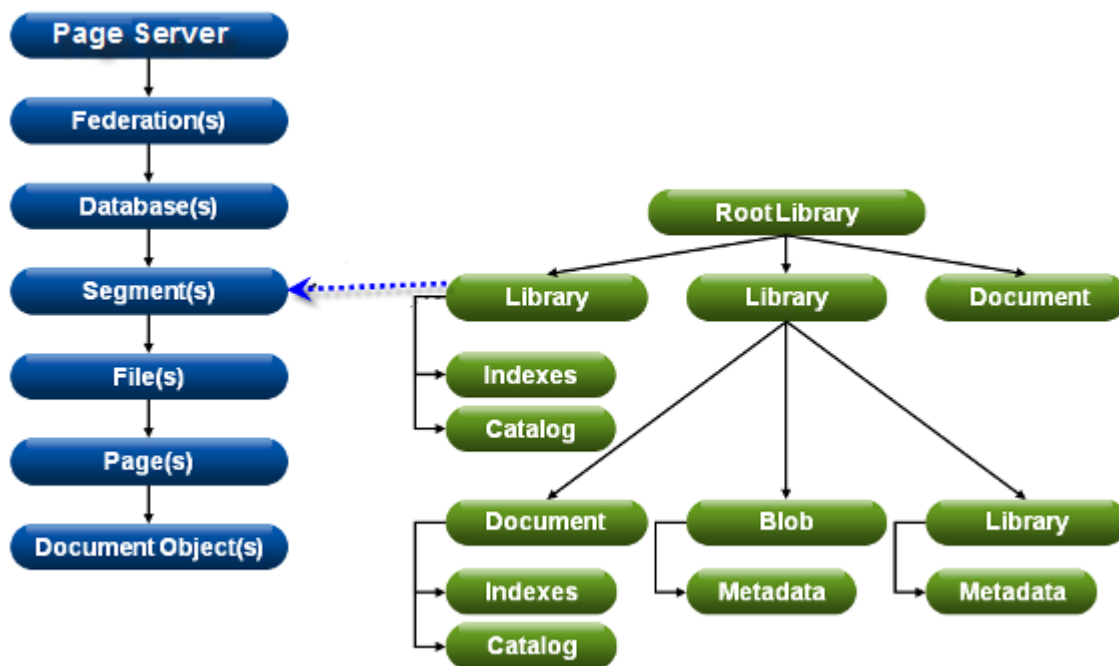
Abstract schemas contain interfaces for handling schema information, such as the structure of element declarations, and interfaces for applying schema information to DOMs validation. There is full abstract schema support for DTDs and a more limited support for XML schema, with some product-specific modifications.

BLOBs

Binary large objects (BLOBs) are binary non-XML files, such as image files (GIF, JPEG, PNG, BMP), sound files (MP3, WAV), and Microsoft Office files (DOC, XLS, PPT). Storing BLOBs and XML documents in the same database allows managing all resources for a specific project or product in one uniform way.

Internal structure: databases, segments, files and pages

Physically, a database consists of one or more segments. Each segment has one or more files and each file occupies one or more pages. The physical and logical structures relate as follows:



A segment is a logical storage location within a database. Each database always has at least one segment, the *default segment*. Segments can be added and empty segments can be deleted. The default segment can never be deleted without deleting the database.

New libraries and data are stored in the default segment, unless specified otherwise. Data is never automatically stored in another segment, even if the current segment is full. If you want automatic overflow, use a single segment with multiple files.

Note: Transactions, even read-only ones, can require *temporary data*, for example new nodes in XQuery queries, or old versions of documents. Unless specified otherwise, such data is held in a *temporary segment*.

Methods exist for attaching a library to an existing segment, for assigning a segment for temporary data and for storing all children of a specific library on a specific segment.

All segments of one library must have the same state at any time.

Database files

A segment can be spread physically over multiple files. A segment always has at least one file, the default file. The default file can never be deleted without deleting its owner segment.

Files can be added to a database segment using the Admin Client.

The maximum size limit of a file can be set when the file is added to the segment, or later, provided the limit is not less than the current size of the file. If a file exceeds the size limit, the overflow data is allocated to the next file of the segment. If all the files in the segment have reached their limits, any further allocation in the segment fails. Allocation is random, and different pages of a single document can be stored in different files. Use different segments if you want to control where the data is stored.

Note: To keep data consistent, xDB sometimes allocates pages for internal page allocation administration. In such cases, xDB ignores the maximum file size limit. As a result, the file may slightly exceed the size limit set by the administrator.

Overflow to another file can be prevented by setting the maximum file size limit to 0 (zero), which effectively means there is no maximum file size. In practice, setting an unlimited file size is useful only for a last file.

Note: A file that has no maximum size may outgrow the available space. Preventing this can affect performance, because the only reliable way to check for a full disk is by actually writing the (still empty) page to the file while allocating the page.

Database configurations

Initial database configuration is specified by a [database configuration file, page 230](#). The default database configuration has a single segment with a single file and all data clustered in the default segment. The Admin Client can be used to change the default configuration, to create or delete segments and files, and to modify default cluster rules.

Detachable libraries

Detachable libraries can be detached from and attached to the database. Once a library is detached, it is not accessible from the database.

A library can be detached if the library and its descendants are stored on a set of segments that are not shared with any other libraries, and the ancestors of the library do not have other indexes than id and name indexes.

A detachable library can have its own MultiPath indexes. The MultiPath index of a detachable library can be merged with descendant library levels, but not with parent or ancestor levels.

A detachable library can have the following mutually exclusive states:

- read-write - Both read and write operations are allowed.
- read-only - Only read operations are allowed.
- detached - The library is logically or physically removed from the database and is not accessible from the database.
- detach-point - Similar state as the detached state, except only a detachable library in a detach-point state can be attached to the database. A detachable library in detached state cannot be directly attached to the database.

By default, a detachable library has a read-write status.

Detachable libraries can be marked as non-searchable. A non-searchable detachable library is not visible to search queries.

Detachable libraries can be backed up and restored individually.

Managing users and groups

xDB provides authorization and security support for managing user actions and user access to information.

The xDB security model is based on three levels: user, administrator, and superuser. These user types have different access levels, as follows:

- Users can access all data for which they have authorization.
- An administrator can access all the data and user information in one database.
- The superuser can create and delete a database, but cannot access any of the data stored in the databases.

The following permission settings can be applied:

- Executable - Specifies that the object can be executed. Currently this setting is not applied to any object in the xDB database.
- Writable - Specifies that the object can be modified or deleted.
- Readable - Specifies that the object can be viewed.

Permissions can be set for the following user types:

- Owner - The user who owns the object.
- Group - The group that has group access rights to the object.
- Other - All other users.

Installing xDB

This chapter contains the following topics:

- **Pre-installation requirements**
- **Installing xDB on a Windows platform**
- **Upgrading xDB on Windows**
- **Installing xDB on a UNIX platform**
- **Upgrading xDB on UNIX**
- **Uninstalling xDB**
- **Verifying the xDB installation**
- **Creating a sample database**

Pre-installation requirements

xDB installers are available for Windows 2000/XP/Vista/7 and for Unix. xDB can be used with virtualized versions of these operating systems running in any version of VMware. Unix installers have been tested on Linux, Solaris, AIX, and Mac OS X, including 64-bit versions of Red Hat Enterprise Linux edition 6.3, AIX V6.1 TL1, Oracle Solaris 10, and HP-UX 11i v3 Itanium.

Before installing xDB, verify that your system meets the hardware and software requirements for xDB.

We highly recommend that you read the readme.txt file that is provided with the xDB distribution.

If your new xDB installation is intended for a non-standard environment or a specific software application, like EMC Documentum Content Server with enabled XML Store or EMC Documentum xPlore, consult the appropriate documentation for possible differences in requirements, installation procedure and configuration.

If you have previous versions of xDB running, you should consider upgrading or uninstalling them. If you want to have multiple xDB installations on one system, for example xDB 9.0 and 10.0 side by side, you must use different installation locations and port numbers for each one.

Note: On Windows, the installer can not install multiple Windows Services for xDB. For example, it can not configure both an xDB 10 **and** an xDB 9 background service.

Note: The xDB installation layout has changed slightly since version 9: third party jar files are now in subdirectories within subdirectory lib, xdb.properties is in subdirectory conf, and the page server will write logs to subdirectory log.

Note: xDB is a Java application, and requires a Sun Java Development Kit 7 or higher, or an SDK with a fully compatible Java Virtual Machine. Ensure that this is present on the target machine *before* you install xDB.

Using the Java command line requires configuring the CLASSPATH variable to include the required [JAR Files, page 68](#).

Table 2 Minimum system requirements for xDB

Java	JDK 7 or higher (http://java.sun.com)
RAM	256MB
Hard drive space for xDB software	100MB
Hard drive space per database	1MB
Network	TCP/IP

Installation parameters

The xDB installer will ask you to provide some parameters, including:

- The location where you want to install xDB.
- The base directory path to the JDK.
- A valid xDB license key. **Note:** xDB software licenses are valid for a limited time. If in doubt about your license, contact xDB customer support.
- An xDB superuser password. This password is required for creating databases and some other administrative tasks.
- Optionally, set [advanced parameters, page 54](#).

Related links

[Installing xDB on a Windows platform](#)

[Installing xDB on a UNIX platform](#)

Installing xDB on a Windows platform

When installing or uninstalling xDB on Windows, you must be logged on as a Windows administrator, or as a user with similar access privileges.

Running `xdb_setup.exe` installs xDB on a Windows machine. The installer copies files to the proper directories, configures the xDB installation, and augments the PATH environment variable.

Note: The installer also sets up a dedicated page server as a Windows Service for the latest xDB installation. It does not allow for multiple Windows Services for xDB. For example, after you install xDB 10 alongside an existing xDB 9, there will be no longer be a Windows Service for the older installation. For more information, see [The xDB dedicated page server, page 70](#).

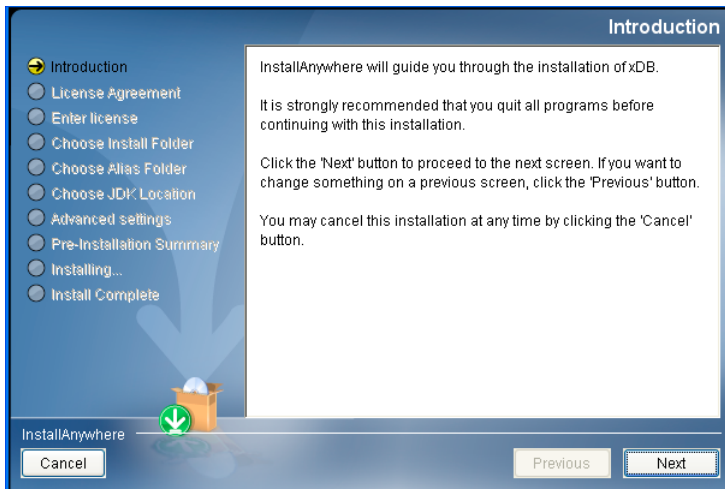
Note: If you are upgrading a previous xDB version, see [Upgrading xDB on Windows, page 57](#).

To install xDB on a Windows platform:

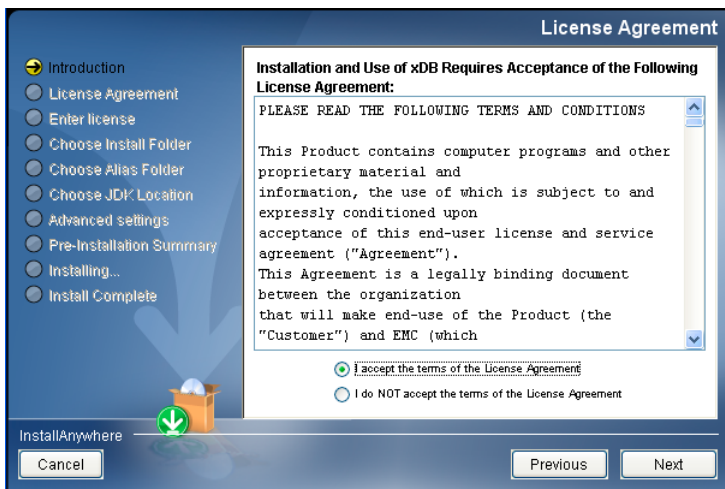
1. Double-click the `xdb_setup.exe` file, located in the root directory of the xDB distribution.

If you want to obtain debug output from the installer, hold down the Ctrl key while running `xdb_setup.exe`, until a console window opens for display of debug output. You can preserve the debug output by copy/pasting it from the console window to a text file **before** you exit the installer.

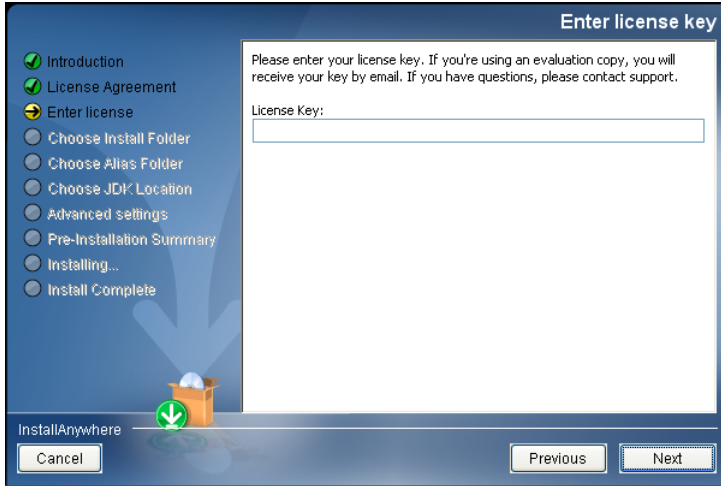
The installer starts the xDB installation, and displays the Introduction. The left panel of the installation window shows the sequence and progress of the installation steps. Use the buttons at bottom right for navigation.



2. Read the introduction, and Click **Next**.
The License Agreement appears.



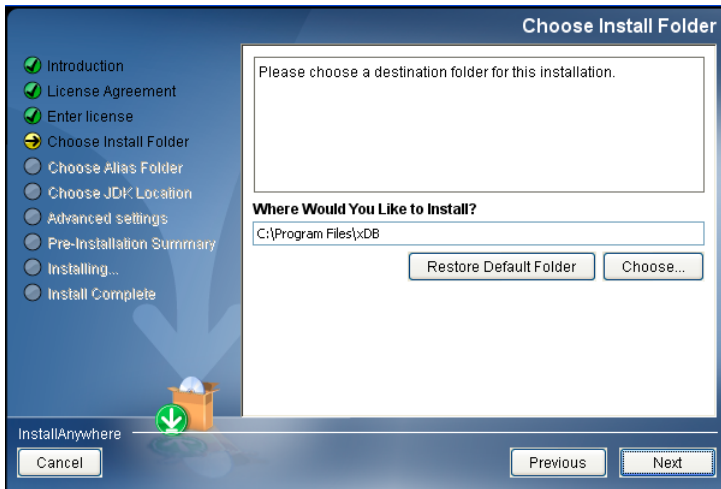
3. Select the **I accept** option for the terms of the software license agreement, then click **Next**.
The Enter license key window appears.



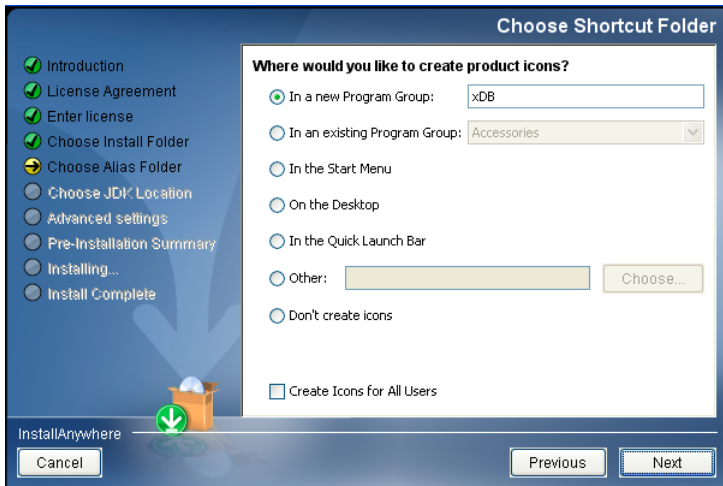
4. Enter a valid xDB license key, then click **Next**.

The Choose Install Folder window appears. The default folder is C:\Program Files\xDB or similar, depending on the locale of your Microsoft Windows installation.

Note: The database files and transaction log files will also be stored here, unless you change their locations from the installation defaults in [Advanced Settings](#).

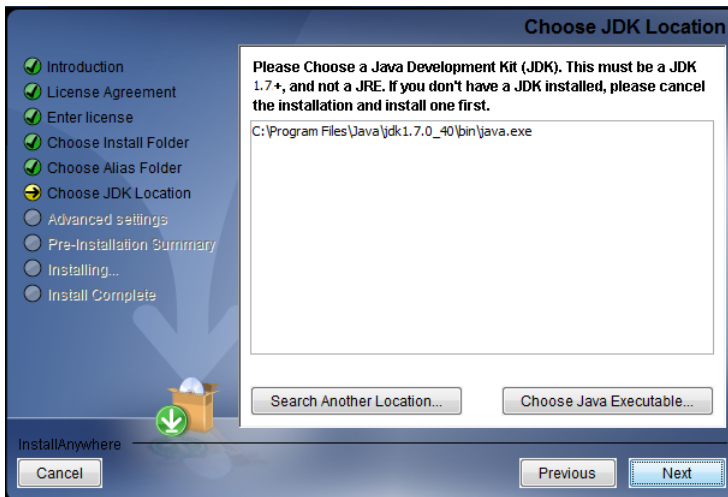


5. Choose a folder or type a folder path for the target installation folder, then click **Next**. The Choose Shortcut Folder window appears.

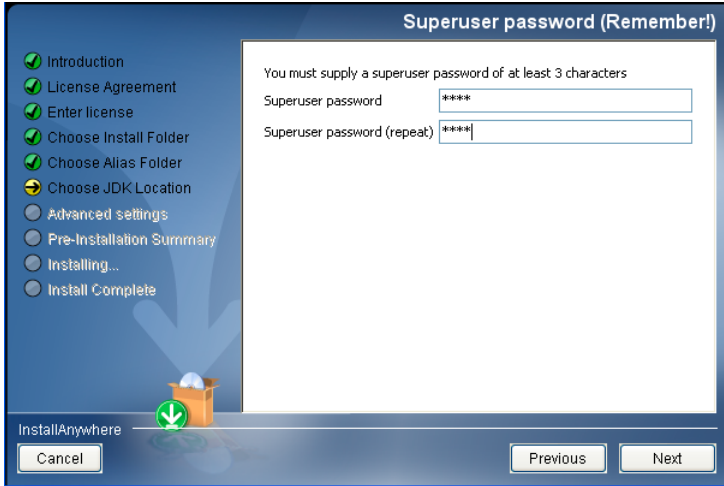


6. Choose where shortcuts for xDB should be created by the installer, then click **Next**.

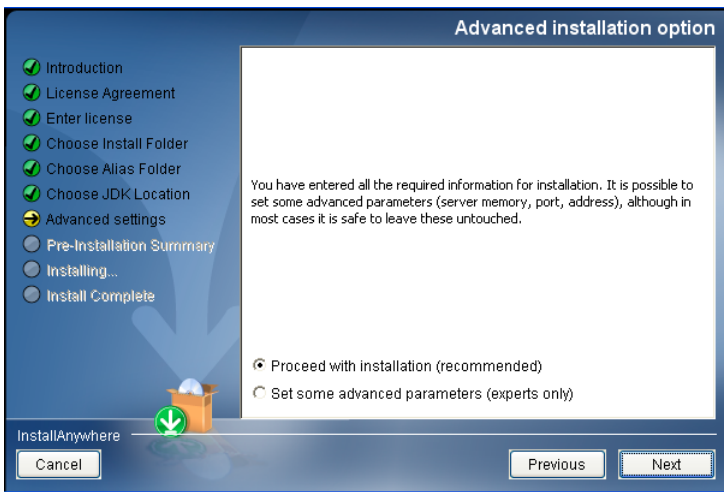
The Choose JDK Location window displays a list of Java executables found on your system. You can you to pick one from the list, or use the **Choose Java Executable** button or the **Search Another Location** button to find the applicable Java Development Kit and set the path to the **java.exe** file.



7. Select the appropriate Java executable, then click **Next**
The Superuser password window appears (see [Superuser](#), page 42).

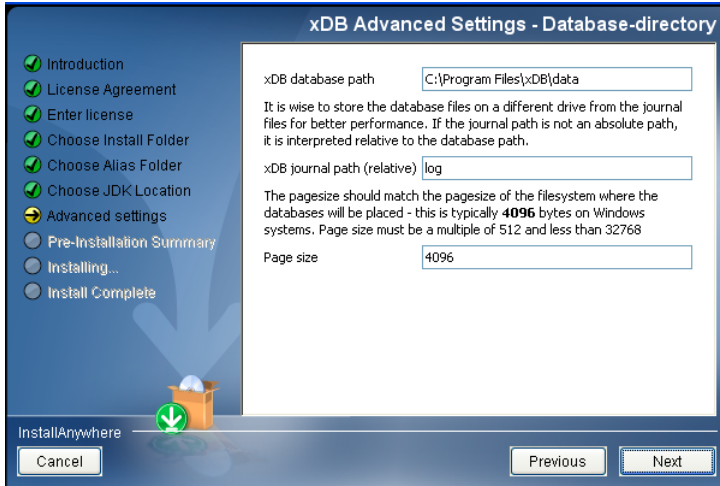


8. Enter the password of the superuser twice, then click **Next**. The Advanced installation option window appears.



9. Specify whether or not to set additional advanced parameters, then click **Next**.
 - To complete the xDB installation with default values for the advanced parameters, select **Proceed with installation**. **Note:** The default storage locations for database files and transaction journal files are in subfolders of the installation folder.
 - To manually change xDB Advanced settings, including paths for database files and transaction journal files, and JVM and xDB server parameters, select **Set some advanced parameters**.

If you selected **Set some advanced parameters**, the xDB Advanced Settings - Database-directory window appears.

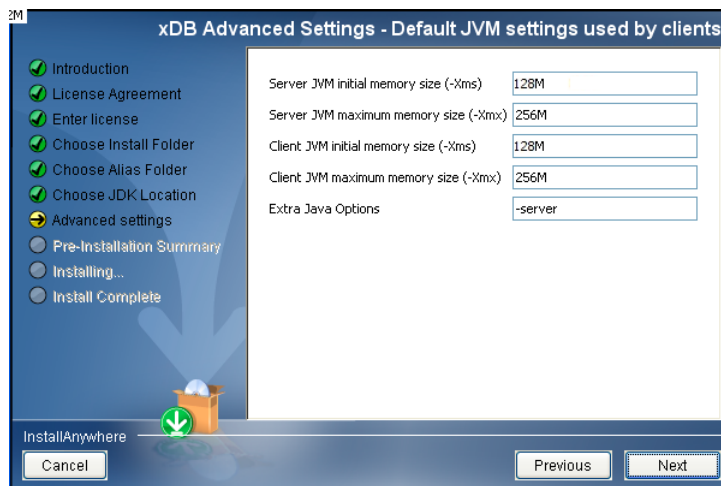


- a. Specify the following:

Advanced Settings	Description
xDB database path	The path to the database files. The default path is based on the installation directory.
xDB journal path	The default path to the transaction log files. For best performance, store the transaction log files on a different physical disk than database files.
Page size	The database page size should match the page size of the filesystem where the database shall reside. See Specifying the database page size, page 77 for more information.

- b. Click **Next**.

The xDB Advanced Settings - Default JVM settings window appears (showing related java command line parameters in parenthesis).

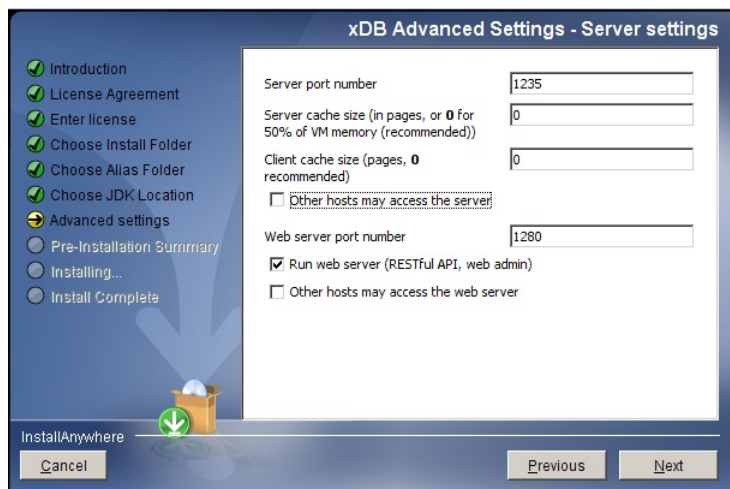


- c. You can specify the following default JVM settings:

Advanced Setting	Description
Server JVM initial memory size (-Xms)	The initial amount of memory to allocate for the server JVM.
Server JVM maximum memory size (-Xmx)	The maximum amount of memory to allocate for the server JVM.
Client JVM initial memory size (-Xms)	The initial amount of memory to allocate for client JVMs. This is used for the Admin client, Java samples, and the command-line client.
Client JVM maximum memory size (-Xmx)	The maximum amount of memory to allocate for client JVMs. This is used for the Admin client, Java samples, and the command line client.
Extra Java Options	Additional options for the JVM.

- d. Click **Next**.

The xDB Advanced Settings - Server settings window appears.



- e. Specify the server port number, and the server and client cache sizes. Optionally, you can allow access by other hosts, and run the internal web server that is required for the xDB [web client](#), [page 246](#).

The Windows installer creates and starts a Windows Service that acts as an xDB page server. The following settings apply to this service:

Server settings	Description
Server port number	The port number of the page server. The page server can run on any available port. A check is made whether the port is available. Note The port number determines the URL which must be used to access the database.

Server settings	Description
Server cache size	The cache size determines the number of pages cached by the server to improve performance. The default value is 0, which is the recommended value. With a cache size value of 0, the database uses half of the JVM memory as cache.
Client cache size	Clients connecting to the server, such as the Admin client or the command-line client, will create their own, local database page cache. This number controls the size of that cache. Half of the JVMs memory ("0") is recommend.
Other hosts may access this server	Select this option if you want to allow processes on multiple machines to access the page server. For security reasons, by default, only processes running on the same machine can access the page server.
Web server port number	The port number of the internal web server.
Run web server	Enables running of the web server, which is required for the xDB web client, page 246.
Other hosts may access the web server	Select this option if you want to allow processes on multiple machines to access the web server. For security reasons, by default, only processes running on the same machine can access the web server.

- f. Click **Next** to finish entering installation settings.

The Pre-Installation Summary appears.

10. Verify your installation settings and make sure that the available disk space is sufficient.
- If not satisfied, use the **Previous** button to go back and change your installation choices.
 - If satisfied, click the **Install** button to proceed with the xDB installation.

The installer shows installation progress. After successful installation, the Install Complete window displays the location of the installation. **Note** Complete installation requires a Windows restart.

11. Set options to display documentation or to Restart Windows, and click **Done**.

Note: On Windows XP or Windows 2008R1, if you wish to use the complete Windows command line functionality, install the appropriate Microsoft Visual C++ redistribution package, downloadable from:

- <http://www.microsoft.com/en-us/download/details.aspx?id=5582> (Windows 32 bit)
- <http://www.microsoft.com/en-us/download/details.aspx?id=2092> (Windows 64 bit)

Upgrading xDB on Windows

On Windows, installers for xDB 10.0 and higher can automatically upgrade an existing installation of xDB version 9.0 or later, provided that you are prepared to give the installer the location and

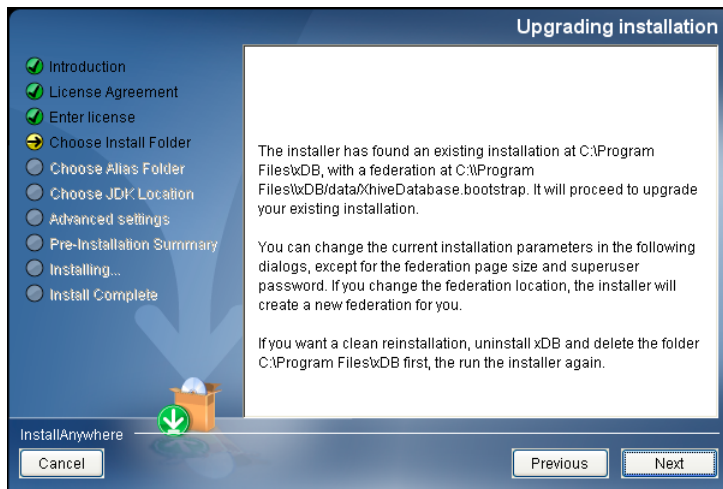
superuser password of the existing installation. The installer needs those for removing the old binaries, background process, documentation, and so on.

Note: Before upgrading, ensure that you have a backup of your existing installation.

1. Execute **xdb_setup.exe** from the distribution, and proceed as for a new installation, until the installer asks for the install location.
2. When asked for the installation location, choose the location of the existing xDB installation that you want to upgrade.

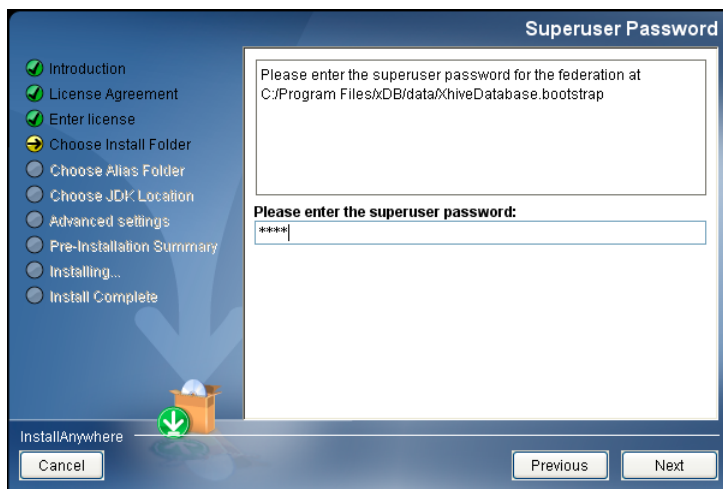
Note: If you do not specify a valid existing xDB location, the installer will create a new installation, instead of upgrading an old one.

The installer displays an upgrade notice for the existing installation, for example:



3. Read the notice.
4. To upgrade the existing installation, click **Next**.

The installer asks for the superuser password for the existing federation, for example:



5. Enter the current superuser password.

Note: To ensure successful installation, the supplied superuser password must be valid and correct.
6. Click **Next**.

The installation process will proceed as for a new installation.

Compatible settings from the previous installation will be adopted automatically. You can change any of the advanced settings during the advanced installation process, except for the page size and superuser password.

Installing xDB on a UNIX platform

On UNIX platforms, xDB can be installed as any user. You need Write permissions in both the installation directories, the directory to which you will untar the distribution and the directory where you will create your initial federation.

You must have a working Java executable in your PATH. You can verify the Java version by running `java -version` from the command line. See also the [Pre-installation requirements, page 49](#).

The xDB installer copies files to the proper directories, configures the xDB installation, and augments the PATH environment variable.

Note: If you are upgrading a previous xDB version, see [Upgrading xDB on UNIX, page 61](#).

To install xDB on a UNIX platform:

1. Extract the distribution `.tar.gz` file to the directory where you want to install xDB.
2. Run the `sh setup.sh setup` script, and enter the parameters that are required during installation.

To obtain debug output from the installer, set environment variable `LAX_DEBUG=true` or `LAX_DEBUG=file` before you launch the installer. The `file` option redirects debug output to a file `jx.log` in the install directory.

```
This script will set up xDB for use in /home/xDB10.5.0 and:
    * create an xdb.properties with default settings for the server and clients
    * create an initial empty federation
```

```
After installation, you can review xdb.properties and adjust it as necessary
Please enter your Java home-directory [/opt/oracle-jdk-bin-1.7.0.17] :
Please enter your xDB license key: ...
Please enter a superuser password      :
Please enter a superuser password (confirm):
Database directory [/home/xDB10.5.0/data] :
Do you want to alter advanced settings? [y/N] [no] : y
Journal files directory (if relative, relative to data directory) [log] :
Page size in bytes [4096] :
Server port number [1235] :
Want webserver running? [y/N] [no] : y
Webserver port number [1280] :
May other hosts access the server? [y/N] [no] : y
Server JVM initial memory size (-Xms) [128M] :
Server JVM maximum memory size (-Xmx) [256M] :
Number of cachepages (0 means 50% VM memory): [0] :
Client JVM initial memory size (-Xms) [128M] :
Client JVM maximum memory size (-Xmx) [256M] :
Number of cachepages (0 means 50% VM memory): [0] :
```

Extra Java Options [-server] :

For most parameters, the xDB installer displays a default value between brackets. For questions that require a Yes or No answer, the default choice is shown in upper case. For example, if the choice is [y/N], N is the default value.

To accept a default, press the `Enter` key. To override a default, type a new value and press `Enter`. If the value is incorrect, a message appears and the question is repeated.

3. Select whether you want to alter advanced settings, as follows:

- If you select `no`, the installation is completed using default settings.
- If you select `yes`, you can modify the following parameters:
 - The directory for journal files. For performance reasons, place the journal files on a hard disk different from the one with your database files.
 - The page size in bytes. Ideally the page size is equal to or smaller than the block-size of the file system where the database is located. See [page sizes, page 77](#) for more information.
 - The port number on which the xDB page server accepts connections.
 - Whether applications on other machines can access the page server on this machine.
 - The minimum and maximum amount of memory available to the JVM (`-Xms` and `-Xmx`), for both server and client processes. The client values are used in scripts to start up client applications like the administration client and the **xhive-ant** command. The server values are used for the **xdb run-server** command.
 - The number of pages the page server caches. More cached pages result in more memory usage but better performance. The default value is 0, which means the server uses 50 percent of the available JVM memory.
 - The number of pages that database clients cache locally in client memory.
 - Other Java command-line options. Usually these options are not changed.

The installer will save settings to the configuration file `conf/xdb.properties` and create an empty federation at the database path you have chosen.

4. Perform the following post-installation steps:

- a. Add the `$XHIVE_HOME/bin` directory to your path, for example:

```
bash$ export PATH="${PATH}:%XHIVE_HOME/bin"
```

or, if you use a (t)csh:

```
tcsh> setenv PATH ${PATH}:%XHIVE_HOME/bin
```

- b. Run the **xdb run-server** command to start the page server.
This command starts a process for the page server, which can then be accessed from the xDB administration tool and command line tool.
- c. Add completion support for the **xdb** command to your shell. See the included `readme.txt` for more details.

Upgrading xDB on UNIX

On UNIX, the xDB installer does NOT upgrade any existing xDB version. If you have a previous version, you must upgrade manually: use a different installation location for the new version, and restore a good backup of the old installation into the new one.

For information on creating and restoring xDB backups, refer to [Creating and restoring backups, page 262](#).

1. Ensure full backup(s) of your old, existing xDB installation(s).
2. Perform a new installation, as described in [Installing xDB on a UNIX platform, page 59](#).
3. Restore your backup(s) to the new installation.
4. Test the new installation.
5. Shut down the old installation, and deploy the new installation for use.

Uninstalling xDB

Depending on the platform on which xDB is installed, the uninstall process is as follows:

- On Windows:

Use the **Uninstall xDB** option in the xDB section of the Windows Start Programs menu to uninstall xDB. The Windows uninstall process also unregisters the dedicated xDB page server service. If you have multiple xDB installations on the same machine, the **Add/Remove Programs** option in the Windows Control Panel applies only to the latest xDB installation.

The uninstall process may not be able to delete all data directories, compiled samples, and other files that were created after xDB was installed. Remaining directories and files can be removed manually afterwards.

- On UNIX:

If the page server is running, use the **xdb stop-server** command to stop it. Then remove the installation and data directories.

Note: The `uninstalling.txt` file contains plain text instructions for uninstalling xDB.

Verifying the xDB installation

After xDB installation, you can use the xDB Admin Client to visually inspect and verify the installation result, and to try out various functions, like [creating a sample database, page 62](#).

You can also try the xDB [command line client, page 246](#). Some of its most important commands are listed in [xDB commands, page 62](#). To display information about a command, run it without any parameters.

Table 6 Important xDB commands

Command	Description
xdb admin	Starts the Admin Client , a tool for maintenance of federations, databases, users and content.
xdb create-database , page 251	Creates a new database.
xdb delete-database , page 246	Removes an existing database.
xdb create-federation , page 250	Creates a new empty federation.
xdb configure-federation , page 246	Sets the superuser password and license key on a federation.
xdb backup , page 263	Saves a federation to a backup file.
xdb restore , page 264	Restores a federation from a backup file.
xdb info , page 251	Displays debug information on currently open transactions and their locks.
xdb run-server , page 252	Starts the dedicated server process for the default federation.
xdb stop-server , page 252	Stops the dedicated server process for the default federation.
xdb suspend-diskwrites , page 246	Ensures the federation files are flushed to disk, and suspends or resumes writing.

Creating a sample database

The xDB installer creates an empty federation. After installing xDB, you can create a sample "united_nations" database. This database is used in some examples discussed in the xDB documentation, and you can also use it to become familiar with xDB.

To create a sample database using the xDB Admin Client:

1. Start the Admin Client.

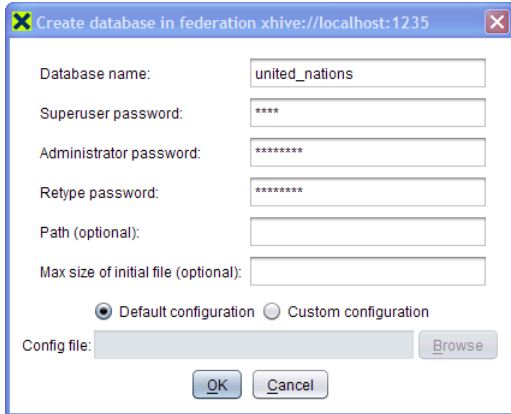
From the command line, you can use the `xdb admin` command in the `\bin` subdirectory of the xDB installation. On Windows, the xDB Admin Client can usually be run from the Windows Start menu.

2. Select **Database > Create database** from the main menu, or type `Ctrl + R`.

The Create database window opens.

3. Enter database name `united_nations` with the superuser password as entered during the xDB installation. Enter `northsea` as the administrator password.

Note: This database name and these passwords are used in all code samples. If you want to use different ones, you must make appropriate changes to the `SampleProperties.java` file.



Dialog box titled "Create database in federation xhive://localhost:1235".

Fields:

- Database name: united_nations
- Superuser password: ****
- Administrator password: *****
- Retype password: *****
- Path (optional):
- Max size of initial file (optional):

Configuration options:

- Default configuration
- Custom configuration

Config file: [] Browse

Buttons: OK, Cancel

4. Click **OK**.

Note: Databases can also be created using the `xdb create-database` command. For more information about using the command line, see [Using the command line client, page 246](#).

Configuring xDB

This chapter contains the following topics:

- **Configuration files for Windows**
- **xDB JAR files**
- **Using the `xhive.bootstrap` property**
- **The xDB dedicated page server**
- **Using external editors with FTP**
- **Enabling FIPS 140-2 Level 1 Encryption**

The xdb.properties file

The xdb.properties file contains properties (key/value pairs) for command-line settings and page server settings. The tables below describe available options.

Command-line settings

The *command-line settings* apply to tools like the **xdb** command and the **xhive-ant** command, as well as to the page server. They can be overridden by setting environment variables with the same names. Alternatively, the corresponding command-line switch(es) can be passed to the tool.

By default, the command-line settings are in the file xdb/conf/xdb.properties.

Optionally, user-specific properties can be set in the user's home directory in a file .xdb.properties (note the leading "." dot).

Note: Changes to server-related settings, such as the XHIVE_SERVER_MAX_MEMORY property, require restarting the server to take effect. Other settings are applied the next time a tool is run.

Table 7 Command-line property settings

Property	Description
XHIVE_BOOTSTRAP	The URL used by command line tools, for example xhive://localhost:1235.
XHIVE_DATABASE	The default database to use. This is not set by the installer.
XHIVE_USERNAME XHIVE_PASSWORD	Default credentials (user name and password) for the command line tools. Default credentials are not set by the installer, and should NOT be set on production systems.
XHIVE_MAX_MEMORY	Maximum memory used by a single command line tool, as in the <code>-Xmx</code> parameter to the JVM.
XHIVE_MIN_MEMORY	Minimum memory used by a single command line tool, as in the <code>-Xms</code> parameter to the JVM.
XHIVE_CACHEPAGES	The number of cache pages allocated to command line tools. If set to 0, half of the JVM memory is used.
XHIVE_FEDERATION	The location of the default bootstrap file. Commands that use this property include xdb run-server , xdb create-federation , and xdb restore .
XHIVE_SERVER_MAX_MEMORY	Maximum memory for the page server process (<code>-Xmx</code>).
XHIVE_SERVER_MIN_MEMORY	Minimum memory for the page server process (<code>-Xms</code>).
XHIVE_OPTS	Additional options to be passed to the JVM.
XHIVE_SERVER_ADDRESS	The page server listens at this address. If set to "*", the server accepts all connections. If set to "localhost", only local connections are accepted.
XHIVE_SERVER_PORT	The port used by the page server.

Property	Description
XHIVE_SERVER_CACHEPAGES	Cache pages for the page server process. If set to 0, half of the JVM memory is used.
XHIVE_HOME	The installation location. It can be changed to use a different software version or an installation in a different location. If left empty, the tools try to infer a location.
XHIVE_JAVA_HOME	The JDK installation to be used with the tools to which the properties apply. This must be a proper Java Development Kit, not a Java Runtime Environment (JRE). If left empty, the tools use the JAVA_HOME path or any java executable on the path.
XHIVE_STATISTICS_MONITORING_ENABLED	If set to true, enables statistics monitoring.
XHIVE_STATS_MONITOR_INTERVAL	The statistics monitoring interval.
XHIVE_WEBSERVER_ADDRESS	The web server listens at this address. If set to "*", the server accepts all connections. If set to "localhost", only local connections are accepted.
XHIVE_WEBSERVER_PORT	The port used by the web server. If left empty, the web server will not run.
XHIVE_WEBSERVER_NO_WEBADMIN	If set to true, the web server will not serve the web admin application.

Page server settings

These property settings apply to the page server (both internal and dedicated).

The page server looks for the xdb.properties file in the Java classpath.

Note: The **xdb** command adds the directory XHIVE_HOME/conf to the Java classpath automatically.

Table 8 Page server settings

Property	Description
XHIVE_FIPS_ENABLED	Enables (true) or disables (false/not set) FIPS 140-2 Level 1 encryption of user passwords. For more information, see Enabling FIPS 140-2 Level 1 encryption, page 73 .
xdb.lucene.*	Various properties related to the multi-path indexes - see Multi-path index properties, page 157 .

Configuration files for Windows

The Windows installer creates lax configuration files for property settings for executables of the [Admin Client, page 227](#) and for the Windows Service of the [page server, page 70](#). For more information about property settings, refer to [the xdb.properties file, page 66](#).

- The file `$XHIVE_HOME\bin\xDB Admin Client.lax` contains options for the Admin Client executable **xDB Admin Client.exe**, that only apply when the Admin Client is run from the Windows Start Menu shortcut or from the executable. **Note:** These options do **not** apply to the **xdb admin** command and the **xhive-ant run-admin** command.
- The file `$XHIVE_HOME\bin\xDB Server.lax` contains the following JVM options for the Windows Service executable **xDB Server.exe**:
 - `lax.nl.java.option.java.heap.size.initial` (equivalent to `-Xms`).
 - `lax.nl.java.option.java.heap.size.max` (equivalent to `-Xmx`).
 - `lax.nl.java.option.additional` (this option changes JVM parameters).
 - `lax.nl.current.vm` (this option changes the default JVM).

xDB JAR files

Table [Required JAR files](#) shows JAR files that are needed for various xDB functionality. For use with the Java command line, the relevant JAR files need to be included in your `CLASSPATH`.

The file `lib/versions.txt` of the xDB distribution contains a list of third-party software with version numbers and licenses.

Table 9 Required JAR files

Functionality	JAR files	Comments
Basic xDB	xhive.jar antlr-runtime.jar, aspectjrt.jar, fastutil-shrunked.jar, guava.jar, icu4j.jar, lucene.jar, lucene-queries.jar, lucene-queryparser.jar, lucene-analyzers-common.jar, lucene-sandbox.jar, xercesImpl.jar, xml-apis.jar	
XSLT transformations	xalan.jar, serializer.jar	If desired, another JAXP compliant XSLT processor can be used, such as Saxon.
PDF transformations	fop.jar, avalon-framework.jar, batik-all.jar, commons-io.jar, commons-logging.jar, xmlgraphics- commons.jar	

Functionality	JAR files	Comments
Command line clients	jline.jar	Only used by the command line clients. Not necessary for applications using xDB.
xhive-ant script	ant.jar, ant-launcher.jar	Only used by the xhive-ant script. Not necessary to run applications.
Spring applications	aopalliance.jar, spring-aop.jar, spring-beans.jar, spring-context.jar, spring-core.jar, spring-expression.jar, spring-tx.jar, spring-web.jar, spring-webmvc.jar	These JAR files are used for the Spring code examples in src\samples\spring.
Webclient	xdb-rest.jar, xdb-webadmin.jar jetty-continuation.jar, jetty-http.jar, jetty-io.jar, jetty-security.jar, jetty-server.jar, jetty-servlet.jar, jetty-util.jar, jetty-webapp.jar, jetty-xml.jar, servlet-api.jar asm.jar, jackson-core-asl.jar, jackson-jaxrs.jar, jackson-mapper-asl.jar, jersey-client.jar, jersey-core.jar, jersey-json.jar, jersey-server.jar, jersey-servlet.jar, jettison.jar	Only used by the command line clients and the Windows Service.
FIPS 140-2 encryption, page 73	cryptoFIPS.jar	FIPS 140-2 encryption requires the RSA BSAFE Crypto-J toolkit version 5.0.1. The toolkit is not shipped with xDB and must be purchased separately - see www.rsa.com .

Using the xhive.bootstrap property

A bootstrap specifies a connection to a federation, like a connection string in relational databases specifies a data source and the means of connecting to it.

The bootstrap property can be used to connect to a page server in two different ways:

- **Client/server:** if the property is a URL of the form `xhive://hostname:port`, connection will be to a dedicated page server running behind the specified TCP/IP port. It takes the form `xhives://hostname:port` for [SSL encrypted connections, page 283](#).
- **Internal server:** if the property is a path to a file, the page server will run in the current JVM. The bootstrap file is usually called `XhiveDatabase.bootstrap`. Depending on the application, an internal server can be faster than using a remote connection, because communication overhead is avoided. However, keep in mind that there can be only one page server at a time for a specific federation.

The xDB dedicated page server

The standard xDB installation procedure configures the system with a dedicated page server, a small Java program which runs as a background process and to which other applications can connect to obtain access to the federation.

Note: This dedicated page server is not a required part of an xDB deployment, and running xDB with a separate dedicated page server usually is not the best configuration for performance. The reason that xDB is configured with a separate server by default is that this generally makes it more convenient for new users to get started. For production applications where performance is essential it may not be the best choice.

Configuring the dedicated page server

The following server configuration values are determined during xDB installation:

- Port number for accepting connection. The default port number is 1235.
- Page-cache size. The default value is 0, allowing the server to use half of the available JVM memory.
- Addresses that are allowed to connect. The default value is 'localhost', which only allows connections from the same machine. Using '*' allows connections from every machine.

These configuration parameters are set in the xdb.properties file in the conf subdirectory underneath the xDB installation home. The server processes must be stopped and restarted after changing anything in the configuration. Enlarging the cache size requires allocating more memory to the process by configuring the XHIVE_SERVER_MAX_MEMORY parameter in the properties file. The selected page size is stored in the XhiveDatabase.bootstrap file, but cannot be changed after creating a federation.

By default, the dedicated process is not configured for SSL connections. For more information about SSL, see [Using the Secure Socket Layer, page 283](#).

Running a background server process on UNIX

Due to differences in system layouts on UNIX systems, the xDB installer does not automatically configure a background service for the page server. Users can set up their own startup item, typically a shell script in /etc/init.d. The page server can be started as a background process using the typical shell syntax.

Example of running a page server in the background on UNIX

```
#!/bin/sh
xdb run-server \
  --debug --non-interactive \
  >>/var/log/xdb-server.log \
  2>>/var/log/xdb-server-error.log &
echo $! > /var/run/xdb-server.pid
```

The **xdb run-server** command starts a server process for the default federation, to which other processes can connect. The log output of any errors is sent to the console.

The **xdb stop-server** command connects to the server JVM running at the configured bootstrap location and tells the server process to terminate.

For more information about these commands, see [Server-related commands, page 252](#) and [Command-line client global options, page 251](#).

Running without a dedicated server

The page server process can be run inside a Java application, instead of using a dedicated page server. Typically, running the server process inside an application offers better performance, and if necessary the embedding application can accept connections from other applications.

To configure use without a dedicated server:

- Use the `/path/to/XhiveDatabase.bootstrap` path as bootstrap in your application, instead of `xhive://hostname:portname`.
- Ensure that no dedicated server is running, because only one process at a time is allowed access to the federation. On Windows, this is more likely than on Unix, because on Windows the installer can configure a dedicated page server by default. On UNIX, the dedicated server is not started automatically by the installer.

Using external editors with FTP

The xDB distribution includes a code sample of an FTP server implementation, which can be used to enable storage and retrieval of documents in an xDB database by FTP-aware editors and other applications.

For information on FTP use, including possible issues or limitations, consult the relevant documentation of your operating system and your editor or other application.

The xDB FTP server can be started using the `xhive-ant run-ftpserver` command, as follows:

```
xhive-ant run-ftpserver -Ddbname=<DatabaseName>
```

For Windows, the URL for accessing an FTP-server has the format `ftp://localhost/`.

Managing DTDs

Some external XML editors must have access to a DTD to edit an XML document. In xDB, the DTDs are located in a catalog and are associated with an XML document using a doctype declaration.

Associating an XML document with a DTD requires:

- Storing the DTD in the database. DTDs can only be stored in folders that represent catalogs.
- Referencing the DTD in the doctype declaration of the document.

Example

The following example associates a DTD with an XML document.

The default doctype declaration for a new XML document is

```
<!DOCTYPE rootElem PUBLIC "publicId" "systemId">
```

There are two ways to associate a DTD with a document:

- Setting the `publicId` string to match a public ID of a DTD stored in the catalog of the library where the document is stored.
- Setting the `systemId` to the full URL to the DTD as it can be accessed remotely, for example `http://localhost:8080/xhive/xhive-catalog/play.dtd`.

When the file is opened again from the server, the document type declaration is changed to the following format:

```
<!DOCTYPE rootElem PUBLIC "xDB public id" "xhive-catalog/fileName.dtd">
```

Troubleshooting DTDs

Every library has access to a catalog that holds a list of stored DTDs. Documents can refer to the DTDs in this catalog using the public ID in the document type declaration. The catalog is available as a folder in each library, regardless of whether that library has a local catalog or not. The server adds the folder to the list of file and directory names. The folder name is arbitrary and can be changed in the source code. When a document is loaded from the database to an FTP client, the client uses the system ID to look up the DTD. The client can only find the DTD in the database if the system ID is a relative path, using a format like `xhive-catalog/fileName.dtd`. The client then requests the `fileName.dtd` file in the `xhive-catalog` folder relative to the library where the document is stored. The server sends the DTD to the client.

The relative file path can be set when the document is parsed for the first time.

The following information can be helpful when troubleshooting DTDs:

- The DTD is not stored when FTP is used to store a new document. .
- To store a new document with a DTD, the public ID must point to an existing file the catalog or pass a full system ID. If the public ID matches a DTD file in the catalog, the document is linked to that DTD.
- Documents with a linked DTD must have a system ID with a format like `xhive-catalog/fileName.dtd`.
- When documents are parsed into the database, they are parsed or created with certain default parameters, which can be changed in the source code of the FTP server.
- When storing documents on the FTP server, new documents can be created during parsing and existing documents can be replaced.
- When documents or DTDs are stored in the database over FTP, the client sends the document contents as a stream to the server. If the document contains relative paths to files on the client system, such as references to DTDs, the references cannot be resolved and parsing can fail.

- The FTP server can store both XML documents and BLOBs based on the file extension. The file extensions that are associated with XML files can be modified accordingly.
- On systems that already have an FTP-server running, the FTP server cannot run on the default port number 21. The port-constant must be changed to connect the FTP server to another port.

Enabling FIPS 140-2 Level 1 Encryption

Optionally, support is available for FIPS 140-2 Level 1 encryption of user passwords, including regular database users as well as federation superusers. FIPS is a standard that describes U.S. federal government requirements for cryptographic modules used in IT products. Level 1 is the base security level without additional physical security mechanisms.

The RSA BSAFE Crypto-J toolkit is required for [FIPS 140-2](#) support. **Note:** The RSA BSAFE Crypto-J toolkit is not included in the xDB distribution and must be purchased separately - see .

To enable FIPS 140-2 Level 1 encryption:

1. Install the RSA BSAFE Crypto-J toolkit by following the instructions included in the Crypto-J installation package. This includes copying the `cryptojFIPS.jar` file under the JDK's `jre/lib/ext/` directory and registering and configuring the Crypto-J JCE security provider in the file `jre/lib/security/java.security`.
2. Enable FIPS 140-2 encryption in `$XHIVE_HOME/conf/xdb.properties` by setting the `XHIVE_FIPS_ENABLED` property to `true`:

```
XHIVE_FIPS_ENABLED=true
```

When FIPS 140-2 Level 1 encryption is enabled, the Crypto-J FIPS JCE security provider can be used to encrypt user passwords. **Note:** Enabling FIPS 140-2 encryption on an existing federation does not affect existing users. FIPS 140-2 encryption applies only to passwords of newly created users and to password changes of existing users. This means that to encrypt the passwords of existing users in a FIPS 140-2 compliant manner, those users must reset their passwords. Passwords encrypted using a FIPS 140-2 compliant algorithm can co-exist with passwords encrypted using other algorithms.

Optimizing Performance

This chapter contains the following topics:

- **Improving server performance**
- **Configuring JVM and cache pages**
- **Choosing the database page size**
- **Linux file system performance**
- **Using multiple disks**
- **Disabling disk-write caches**
- **RPC tracing**

Improving server performance

Generally, the easiest and most effective way to speed up an application is to run the page server in the same JVM as the application. In this case, client and server communicate via method calls, which is much faster than communicating via TCP/IP. Additionally, the system does not use a separate client and server cache, which allows using more RAM for the single cache.

Because only one page server can run for a specific federation, an internal server can only be used if the application architecture allows for it. If your architecture has multiple users simultaneously running a client application that uses the server directly, you cannot use an internal server in that client application: each application would attempt to start its own page server, and since there can be only one, only the first application would succeed. However, in application architectures where all database accesses are done from a single application server, using an internal server not only speeds up database calls, it also simplifies application deployment.

An internal server can also function as a server for remote clients, provided that the main application contains the necessary java code. If an internal page server is run within an application server, it is still possible to connect to it, for example with the Admin Client, the `xdb backup` command or with a data loading utility. By using the path and name of a bootstrap file as a bootstrap property, the main application can run the page server internally. Any other application can then connect to the server that runs in the main application by using an `xhive://host:port` URL as bootstrap.

Configuring JVM and cache pages

There are several JVM settings that impact the performance of the page server. The easiest to use and most important one is the `-server` flag. Some JRE versions come with a client and a server compiler.

Using the server compiler can improve performance for CPU bound processes significantly. The trade-off is a slower application startup that is irrelevant for server applications.

Upgrading to a new major version of the Sun JDK often causes a 10 percent performance improvement for CPU bound applications due to optimizations. The new versions are more stable as well, and it is recommended to use the latest release.

Depending on the application, the amount of memory available to the JVM and the page server cache can also be important. Generally, the more memory is available, the better the performance. The default installation configuration does not have much impact on a typical developer desktop, but is usually insufficient for real server applications. It is impossible to recommend specific numbers, because the optimal settings will depend on circumstances, including the application and the data, the hardware and any other tasks that the hardware has to perform.

As a first estimation of the required cache pages for a page server, take common operations in the application, such as frequently run XQueries. Make sure the queries are supported by indexes, then add up the size of the cache pages occupied by the indexes. The Admin Client shows the index page sizes on its Indexes tab.

Measure the performance of these common operations and see if they meet performance requirements. Gradually increase the size of the page server cache until it does not increase performance. You can obtain performance statistics in the Admin Client via main menu option **Settings -> Performance Statistics**.

If statistics show that during a certain amount of work the number of cache pages loaded increases, not all operations were serviced from the page cache. If the number of cache pages loaded does not increase for a set of operations, then increasing the cache size will not increase performance.

Cache size does not have an impact on parsing new documents into the database.

To calculate the amount of memory used by the page server cache, multiply the number of cache pages with the page size (in addition, there is a small amount of overhead per page, which should be insignificant). The default value 0 for cache pages will automatically take half of the available JVM heap.

```
# cache pages * page size = total memory
memory for cache / page size = # cache pages
```

For example, if you have a page server server running with a 16 GB heap, and you'd like to allocate 75% of that as a cache on a federation with 4096 byte pages, the calculation would be roughly:

```
(75% * 16 GB) / (4096 bytes) = cache pages
(12 GB) / (4 KB) = cache pages
(12 Million) / (4) = cache pages
3 Million = cache pages
```

Note: Do not allocate the complete heap as a database cache. The server needs some room to maintain additional data structures outside the cache in order to function.

A larger page cache leaves a smaller Java heap for regular program operation, which can cause excessive Garbage Collection. Increasing the page server cache might thus lead to more frequent/longer pauses and/or decrease application throughput. Tuning JVM Garbage Collection is outside of the scope of this article, refer to your JVM supplier's documentation.

Choosing the database page size

Creating a federation requires specifying a page size for its databases. This value should be a power of 2 in the range from 512 to 65536. In most cases, it is best to use the same value for the database page size as the file system uses.

Each document and BLOB occupies an integer number of database pages. A smaller database page size than the file system page size may save disk space, but at a cost in performance: when writing a database page, the operating system must retrieve the old file system page, which involves copying the database page into the file system page and writing back the whole file system page. When database page size equals file system page size, retrieving the old file system page is not necessary.

Note: The database page size should never exceed the file system page size, because then file-writes are not atomic. If the operating system crashes while a database page is only partly written to the disk, that page becomes inaccessible, and it may even corrupt the entire database.

File systems

On Solaris operating systems, the default file system block size is 8192 bytes. On Windows with a default NTFS file system and on Linux operating systems, the file system block size is 4096. These default block sizes can be modified. Below is a list of commands that report block sizes on various operating systems:

- On Windows 2000, **chkdsk** displays the size of an allocation unit (this command also checks the file system, which may take some time to complete).
- On Windows XP and later, **fsutil fsinfo ntfsinfo** displays the number of bytes per cluster.
- On Linux with the ext2/ext3 filesystem, **tune2fs -l /dev/device** displays the block size of the file system. On Linux with xfs, use the **xfs_info /mountpoint** command. The **mount** command displays the mapping between devices and logical mount points.
- On Solaris and HP-UX, **mkfs -m /dev/device** displays the block size (bsize) of the file system. This command may also work on other Unix-variants. Use with care, as mkfs is also used to create new filesystems.

Linux file system performance

On Linux, there are some file systems for disk formatting. The xfs file system provides good throughput on large files, and offers the best performance for typical page server usage. For more information, see the [XFS FAQ](#).

Using multiple disks

If possible, use a separate disk for storing federation [transaction logs, page 43](#). Writing changes to the federation's transaction logs is more performance critical than modifying the [database files, page 46](#). A committing transaction has to wait for its log records to be flushed to the disk before it can continue. Unless the cache has too many dirty pages in it, modifications to database files happen asynchronously from a background thread.

If multiple disks are used to store the data, the easiest and most effective way is a RAID 0 or RAID 1+0 configuration.

Disabling disk-write caches

Most hard disk drives use an internal write cache for buffering. When the operating system writes a block to the disk, the disk confirms the write action as soon as the data is in the drive cache.

On rare occasions, a power failure or similar condition can prevent the data in the cache from actually being written to the drive platters. If the operating system properly requests the disk to flush its cache when an application calls for it, this should not be a problem. If the drive has a backup battery or similar device to guarantee that confirmed writes are always written to the physical disk even on a power failure, there is no issue.

When looking for further ways to prevent data corruption on power failure, consider disabling the write cache on the relevant disk drives (for example, on Linux NFS, by using the "noac" mount option). **Note:** Disabling the write cache can have a significant negative effect on performance.

RPC tracing

When RPC tracing is enabled, all Remote Procedure Calls to the backend server are logged. RPC traces are useful in performance tuning and trouble-shooting, especially in a multi-node configuration.

Note: The section about RPC tracing assumes that the `java.util.logging.SLF4J` binding is used. For more information refer to the section about [message logging](#), page 286.

An RPC call trace message can contain the following items:

1. Begin time
2. Logger name
3. Logging level
4. Thread ID
5. Session ID
6. Transaction ID
7. RPC request type
8. RPC call duration (in milliseconds)
9. Number of bytes sent
10. Number of bytes received
11. Host address of front-end machine
12. Host address of backend server
13. Node name
14. Method name
15. Parameters
16. Return value or exception

The logging mode and message format for RPC call trace messages can be configured in system properties, as described in the table below.

Table 10 RPC tracing mode and format

System property	Description
XHIVE_RPC_TRACING_MODE	<p>The logging mode. Valid values are:</p> <p>standard - includes all 16 items.</p> <p>compact - includes all items except Parameters and Return value.</p> <p>The default mode is standard.</p>
XHIVE_RPC_TRACING_FORMAT	<p>The trace message format. Valid values are:</p> <p>plain - plain text format.</p> <p>xml - XML format.</p> <p>The default format is plain. In plain text format, items within the same trace message are delimited by space.</p>

Example RPC call trace message in plain text format:

```
2009-07-31T15:57:50.262 com.xhive.trace.rpc FINEST thread-1
RemoteSession@15a6029 0 REQUEST_AUTHENTICATE 23 msec 39 bytes sent
13 bytes received 127.0.0.1:1797 127.0.0.1:1794 primary
requestAuthenticate(userName=Administrator,password=*****,
databaseName=MyDatabase,authSession=true,prechecked=false)
```

The same RPC call trace message in XML format:

```
<rpc-trace beginTime="2009-07-31T15:57:52.590"
loggerName="com.xhive.trace.rpc" loggingLevel="FINEST" threadId="main"
sessionId="testXMLFormatSystemProperties1" transactionId="0"
requestType="REQUEST_AUTHENTICATE" duration="0" bytesSent="39"
bytesReceived="13" frontEndAddress="127.0.0.1" frontEndPort="1797"
backEndAddress="127.0.0.1" backEndPort="1794" serverNodeName="primary"
methodName="requestAuthenticate">
<param name="userName">Administration</param>
<param name="password">*****</param>
<param name="databaseName">MyDatabase</param>
<param name="authSession">true</param>
<param name="prechecked">false</param>
</rpc-trace>
```

Enabling or disabling RPC tracing

By default, RPC tracing is disabled. RPC tracing can be turned on or off using the XHIVE_RPC_TRACING_ON system property. To enable RPC tracing use:

```
java.exe -DXHIVE_RPC_TRACING_ON=true
```

Enabling RPC tracing at system level

RPC tracing can be enabled at system level using the `XHIVE_RPC_TRACING_ENABLE` system property. Trace message format and trace mode can be configured using the `XHIVE_RPC_TRACE_FORMAT` and `XHIVE_RPC_TRACE_MODE` properties.

To enable RPC tracing a system level:

1. Configure the message format and trace mode in the `%JAVA_HOME%/jre/lib/logging.properties` file. For descriptions of these system properties, refer to [RPC tracing, page 78](#).
2. Open a console and enable tracing as follows:

```
java.exe -DXHIVE_RPC_TRACING_ENABLE=true
```

To disable tracing, set the `DXHIVE_RPC_TRACING_ENABLE` property to **false**:

```
java.exe -DXHIVE_RPC_TRACING_ENABLE=false
```

Sending RPC trace output to console or file

The `com.xhive.trace.rpc` Java logger and its associated logging handler record RPC trace messages when RPC tracing is initialized.

You can use the `%JAVA_HOME%/jre/lib/logging.properties` file to set the logging level and to direct xDB trace message output to console and/or to a file, as discussed below. **Note:** The xDB `JAVA_HOME` can be set by `XHIVE_JAVA_HOME` in the file `xdb.properties`.

- Set the logging level for the RPC tracing logger.

To set the RPC tracing logging level to **FINEST**:

```
com.xhive.trace.rpc.level = FINEST
```

- Set the handler(s) and the corresponding logging level(s).

For example, to send verbose trace output to console and to file, set the handler(s) to `FileHandler` and `ConsoleHandler` and to set the corresponding logging level for each:

```
handlers= java.util.logging.FileHandler, java.util.logging.ConsoleHandler
```

```
java.util.logging.FileHandler.level = FINEST
```

```
java.util.logging.ConsoleHandler.level = FINEST
```

If you send trace output to file, you can configure the trace file name, max size, and other parameters in the logging.properties filehandler keys, as described in the table [FileHandler configuration keys, page 81](#).

Note: To apply the changes to the Java logging properties file, you must restart the application. Changes to system properties are dynamic.

Table 11 FileHandler configuration keys

Key	Description
java.util.logging.FileHandler.pattern	Specifies a pattern for generating the output file name. For more information, refer to the java.util.logging.FileHandler API Java Doc.
java.util.logging.FileHandler.limit	Specifies an approximate maximum amount in bytes to write to any one file. If the value is 0, there is no limit. The default value is 0.
java.util.logging.FileHandler.count	Specifies how many output files to cycle through. The default value is 1.
java.util.logging.FileHandler.level	Specifies the default level for the Handler. This key must be set to FINEST .
java.util.logging.FileHandler.formatter	Specifies the name of a Formatter class to use. Set this key to <code>com.xhive.trace.log.RPCSimpleFormatter</code> when the value of <code>XHIVE_RPC_TRACING_FORMAT</code> is plain . Set this key to <code>com.xhive.trace.log.RPCXMLFormatter</code> when the value <code>XHIVE_RPC_TRACING_FORMAT</code> is xml .

Refer to <http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html> for details about JDK logging properties, .

Methods for RPC tracing

Enabling or disabling tracing at session level overrides enabling or disabling tracing at application level, which in turn overrides enabling and disabling tracing at system level.

At session (JVM) level, RPC tracing is enabled or disabled using the `enableRPCTracing()` and `disableRPCTracing()` methods of the `com.xhive.core.interfaces.XhiveSessionIf` class. When RPC tracing is enabled or disabled at session level, it is not necessary to set any system properties or logging properties.

For more information about the `enableRPCTracing()` and `disableRPCTracing()` tracing methods, see the Java API documentation.

RPC Trace XML schema example

Below is an example of an XML schema for RPC tracing.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="rpc-trace" type="rpc-trace-type"/>
</xs:schema>
```

```

<xs:complexType name="rpc-trace-type">
  <xs:sequence>
    <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="name" type="xs:string" use="required"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="returnValue" type="xs:string" minOccurs="0"/>
    <xs:element name="exception" type="exception-type" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="beginTime" type="xs:string" use="required"/>
  <xs:attribute name="loggerName" type="xs:string" use="required"/>
  <xs:attribute name="loggingLevel" type="xs:string" use="required"/>
  <xs:attribute name="threadId" type="xs:string" use="required"/>
  <xs:attribute name="sessionId" type="xs:string" use="required"/>
  <xs:attribute name="transactionId" type="xs:long" use="required"/>
  <xs:attribute name="requestType" type="xs:string" use="required"/>
  <xs:attribute name="duration" type="xs:long" use="required"/>
  <xs:attribute name="bytesSent" type="xs:long" use="required"/>
  <xs:attribute name="bytesReceived" type="xs:long" use="required"/>
  <xs:attribute name="frontEndAddress" type="xs:string" use="required"/>
  <xs:attribute name="frontEndPort" type="xs:unsignedShort" use="required"/>
  <xs:attribute name="backEndAddress" type="xs:string" use="required"/>
  <xs:attribute name="backEndPort" type="xs:unsignedShort" use="required"/>
  <xs:attribute name="serverNodeName" type="xs:string" use="required"/>
  <xs:attribute name="methodName" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="exception-type">
  <xs:sequence>
    <xs:element name="message" type="xs:string"/>
    <xs:element name="frame" type="frame-type" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="frame-type">
  <xs:sequence>
    <xs:element name="class" type="xs:string"/>
    <xs:element name="method" type="xs:string"/>
    <xs:element name="line" type="xs:int" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

Creating Applications

This chapter contains the following topics:

- **Building and running applications**
- **Running a sample**
- **Creating a database using the API**
- **Connecting to a database**
- **Getting a database configuration**
- **Using sessions and transactions**
- **Creating libraries**
- **Storing BLOBs**
- **API methods for managing users and groups**
- **Using a RAM segment for temporary data**
- **The xDB dedicated page server program**
- **Using the FederationSet API**
- **Using xDB with Maven 2**
- **Using xDB with Spring**
- **xDB and OSGi**
- **Using xDB with JAAS**
- **Using the API with SSL**

Building and running applications

This chapter provides basic instructions on how to develop xDB applications, dealing with basic tasks such as database creation and connection, programming a transaction, creating libraries, storing BLOBs, managing users and running a dedicated page server. It also discusses xDB support for Maven 2, Spring, OSGi, JAAS and SSL.

Subsequent chapters discuss working with documents, session/transaction management, indexes, querying, administration tools, and more advanced subjects.

The text includes references to code samples that you can run using the xhive-ant tool, provided you have installed xDB and created a sample database, as described in the chapter about [installing xDB](#), page 49.

For information on compiling and running code samples and other programs, see [Running a sample, page 84](#).

Running a sample

A standard xDB installation includes a `/src/` directory, which contains a number of java code samples. Most of the general examples that the manual refers to are in `/src/samples/manual`. These samples can be run using the `xhive-ant` tool:

```
xhive-ant run-sample -Dname=manual.[sample]
```

Note: The settings in the file `SampleProperties.java` must match your current setup. These settings include the superuser and administrator passwords and the database name. **Note:** The `/src/` directory also includes some separate, special samples that need to be set up and run differently from the “manual” Java code samples. For more information, refer to the file `/bin/build.xml` and the inline Javadoc documentation of the sample’s source code.

To run a “manual” sample:

1. Open a command prompt and navigate to the `/bin` directory.
2. Enter a `run-sample` command, for example:

```
xhive-ant run-sample -Dname=manual.StoreDocuments
```

The **xhive-ant** command sets the proper `CLASSPATH` and other parameters. The example command above runs a sample that inserts two documents into the database. If the command runs successfully, a message appears stating the number of documents stored in the database.

Creating a database using the API

You can create a database with the `createDatabase()` method of the `com.xhive.core.interfaces.XhiveFederationIf` interface. To create a database manually, you can use the [Admin Client, page 230](#) or the `xdb create-database` command, [page 251](#).

1. Start a session and open a connection as superuser. The `databaseName` parameter of the `connect()` call should be **null**.

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver();
if (!driver.isInitialized()) {
    driver.init(1024);
}
XhiveSessionIf session = driver.createSession();
session.connect(superUserName, superUserPassword, null);
```

2. Get a handle to the federation, as follows:

```
XhiveFederationIf federation = session.getFederation();
```

3. Call the **createDatabase()** method with the name of the new database and its administrator password.

```
federation.createDatabase(newDbName, administratorPassword, null, System.out);
```

This creates a default configuration. If you want a custom configuration, you can specify a [configuration file, page 86](#). For more information about the physical file structure of a database, see [Database files, page 46](#).

Note: The superuser can create and delete databases, but cannot administer them. To perform administrative actions on the new database, you need to disconnect and then reconnect as database administrator.

Note: The superuser is not represented by an object in the xDB API.

Note: If you changed the default superuser password of the xDB installation, change the source code of the CreateDatabase sample accordingly.

Samples

[CreateDatabase.java](#)

API documentation

[com.xhive.core.interfaces.XhiveFederationIf](#)

Connecting to a database

Applications can explicitly specify a bootstrap by calling **XhiveDriverFactory.getDriver(String bootstrap)**. When called without a parameter, **XhiveDriverFactory.getDriver()** tries to find a federation via (in this order):

- the Java system property `xhive.bootstrap`
- the first line of a text file called `xhive.bootstrap` in the current working directory of the Java process
- the environment variable `XHIVE_BOOTSTRAP` (The xDB command line tool will also use this environment variable, if run without an explicit federation argument.)

Note: To use the same internal xDB server from different applications, those applications must use the same Java class loader to load the xDB classes. Otherwise, the xDB code loaded by each class loader attempts to start its own xDB server and all but the first one fail.

To connect to an xDB database:

1. Obtain an xDB driver:

```
XhiveDriverIf xhiveDriver = XhiveDriverFactory.getDriver();
```

If you did not specify the bootstrap in the JVM environment, pass the location as an argument to the **getDriver()** method, for example:

```
XhiveDriverIf xhiveDriver = XhiveDriverFactory.getDriver("xhive://localhost:1235");
```

If you connect to the database without a server, use a path to the `XhiveDatabase.bootstrap` file.

2. Initialize the local page cache shared by the sessions for this driver:

```
xhiveDriver.init();
```

You need initialize a specific driver only once in your application. You can use the **isInitialized()** method to verify whether the driver has been initialized.

3. Create a new **XhiveSessionIf** session using the **createSession()** method from the **com.xhive.core.interfaces.XhiveSessionIf** interface:

```
XhiveSessionIf session = xhiveDriver.createSession();
```

4. Connect to the database, supplying a user name, password, and database name:

```
session.connect(UserName, UserPassword, DatabaseName);
```

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver();
if (!driver.isInitialized()) {
    driver.init(1024);
}
XhiveSessionIf session = driver.createSession();
session.connect( userName, userPassword, databaseName );
```

Samples

[ConnectDatabase.java](#)

API documentation

[com.xhive.XhiveDriverFactory](#)

[com.xhive.core.interfaces.XhiveDriverIf](#)

[com.xhive.core.interfaces.XhiveSessionIf](#)

Getting a database configuration

The API method `XhiveDatabaseIf.getConfigurationFile()` returns a DOM Document with the database's current [configuration, page 46](#).

API documentation

[com.xhive.core.interfaces.XhiveDatabaseIf.html#getConfigurationFile\(\)](#)

Using sessions and transactions

In xDB, all database operations take place within a session. The developer can determine the scope of a session. One or more transactions can occur within a session. For example, a transaction can be a group of operations that accesses and updates XML documents or parts of XML documents in a database. Uniting a group of operations in a transaction makes that group of operations atomic, ensuring that the database is never left in an inconsistent state: either all the instructions complete successfully or the entire transaction fails.

To performing transactions, first connect to a database and create a session, as described in [Connecting to a database, page 85](#), then follow the steps below. For more information, see the chapter on [Session and Transaction Management, page 133](#).

To use transactions within xDB:

1. Start the transaction with the **begin()** method of the **com.xhive.core.interfaces.XhiveSessionIf** interface.
2. Enter the instructions that should be executed during the transaction.
3. End the transaction with either the **commit()** or the **rollback()** method.

The **commit()** method completes the transaction.

The **rollback()** method reverses all instructions in the transaction. It should always be used if a failure or unexpected exception occurs during the transaction, for the sake of database consistency. For example, if the disk space is exceeded while loading a document, partial document modifications can result in an inconsistent data structure.

Note: After a call to **commit()** or **rollback()**, all references to database objects (such as nodes, libraries, etc.) become invalid. If you want to continue using the objects after a **commit()**, use **checkpoint()** instead. This method commits all persistent operations executed since the previous **begin()** or **checkpoint()** method call. The transaction remains active after the **checkpoint()** call and references to database variables remain usable.

Example

The example transaction below parses external XML documents and appends them to a library. If an error occurs during parsing or appending, the entire transaction is rolled back and none of the documents are appended.

```
session.begin();
try {
    XhiveLSParserIf parser = rootLibrary.createLSParser();
    for ( int i=1; i<=numFiles; i++ ) {
        XhiveDocumentIf newDocument =
            parser.parseURI( new File(baseFileName + i + ".xml").toURL().toString());
        rootLibrary.appendChild(newDocument);
    }
    session.commit();
} catch (Exception e) {
    // in case of an error: do a rollback
    session.rollback();
    e.printStackTrace();
} finally {
    // always ensure that the session is cleaned up in a finally block
    if (session.isOpen()) session.rollback();
    // remove the session
    session.disconnect();
    session.terminate();
}
```

```
}
```

Samples

[UseSessions.java](#)

API documentation

[com.xhive.core.interfaces.XhiveSessionIf](#)

[com.xhive.core.interfaces.XhiveDriverIf](#)

Creating libraries

The nested structure of libraries within a database is like the nested structure of directories or folders within a file system. There is only one root library, which is created automatically with a new database. You can add libraries and documents as needed to build a suitable document hierarchy or storage architecture.

To create a library:

1. Obtain a handle to the parent library. If that parent is the root library, use the **getRoot()** method to get a handle. Otherwise, use a previously instantiated variable.
2. Create the library using the **createLibrary()** method.
3. Select a unique name for the new library using the **setName()** method.

Note: Naming a library is optional, but strongly recommended because several access and indexing methods only work with named libraries.

4. Append the new library to its parent using the **appendChild()** method.

Example

The sample code below creates a library called `Publications` in the root, with one nested library called `General Info`.

```
// get a handle to the root library
XhiveLibraryIf rootLibrary = united_nations_db.getRoot();

// create a library
XhiveLibraryIf newLibA = rootLibrary.createLibrary();

// give the new library a name
newLibA.setName("Publications");

// append the new library to its parent
rootLibrary.appendChild(newLibA);

// create a library which is a sublibrary of newLibA
XhiveLibraryIf newLibA1 = newLibA.createLibrary();

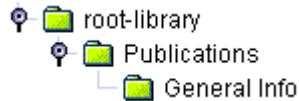
// give the new library a name
newLibA1.setName("General Info");
```



```
// append the new library to its parent
newLibA.appendChild(newLibA1);
```

The sample code creates the following [library hierarchy](#):

Figure 3 Library hierarchy



Related topics

[Managing detachable libraries](#)

Samples

[CreateLibrary.java](#)

API documentation

[com.xhive.core.interfaces.XhiveDatabaseIf](#)

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

[com.xhive.dom.interfaces.XhiveLibraryChildIf](#)

Storing BLOBs

xDB stores BLOBs as a special type of node, the BLOB node. The method `createBlob()` creates a BLOB node. After creating a BLOB node, the content of the node must be filled through the `setContents()` method. To add the BLOB node, the normal methods for adding nodes can be used: `appendChild()` or `insertBefore()`:

```
String imgFileName = "un_flags.gif";
String imgName = "Flags of UN members";
FileInputStream imgFile = new FileInputStream(SampleProperties.baseDir + imgFileName);

// create BLOB node
XhiveBlobNodeIf img = charterLib.createBlob();

// set the contents and name of the BLOB node
img.setContents(imgFile);
img.setName(imgName);

// append the BLOB node to the library
charterLib.appendChild(img);
```

BLOBs stored in xDB can be retrieved using the `getContents()` method in `XhiveBlobNodeIf`. This method returns an **InputStream**. The `getSize()` method returns the size of the BLOB in bytes:

```
// retrieve the contents of the BLOB node
InputStream in = img.getContents();
```

```
// retrieve the size of the BLOB node
int imgSize = (int)img.getSize();
```

A `FileOutputStream` can be used to output the contents of the BLOB node to a file:

```
// output the image to a new file
FileOutputStream out =
    new FileOutputStream(SampleProperties.baseDir + "copy_of_" + imgFileName);
try {
    byte[] buffer = new byte[imgSize];
    int length;
    while((length = in.read(buffer)) != -1) {
        out.write(buffer, 0, length);
    }
} finally {
    out.close();
    in.close();
}
```

When iterating over the child nodes of a library, BLOB nodes can be distinguished from other nodes by their node type which is `XhiveNodeIf.BLOB_NODE`:

```
Node n = charterLib.getFirstChild();
while (n != null) {
    if (n instanceof XhiveBlobNodeIf) {
        System.out.println( "BLOB node found: " + ((XhiveLibraryChildIf)n).getName());
    }
    n = n.getNextSibling();
}
```

Samples

[StoreBLOBs.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveBlobNodeIf](#)

[com.xhive.dom.interfaces.XhiveNodeIf](#)

API methods for managing users and groups

The basic security and authorization levels use the xDB authority object in the `XhiveAuthorityIf` interface. Authority objects are associated with document objects. Each document object always has exactly one attached authority object. An authority object can assign one of the following permission settings to the document object:

API methods for managing users, user lists, groups, and group lists are listed in table [Management interfaces and methods, page 90](#).

Table 12 Management interfaces and methods

Interface	Methods
XhiveUserListIf	addUser(), removeUser(), getUser(), hasUser()
XhiveUserIf	setPassword(), isAdministrator()

Interface	Methods
	addGroup(), removeGroup(), getGroup(), hasGroup() isMember(), users()

Example

The following example uses the **XhiveUserListIf** and **XhiveGroupListIf** interface to create a user, a group, and then adds the user to the group.

```
XhiveUserListIf userList = united_nations_db.getUserList();
XhiveGroupListIf groupList = united_nations_db.getGroupList();

// create a new user (unless it already exists)
if ( !userList.hasUser(userName) ) {
    userList.addUser(userName, userPassword);
}

// create a new group (unless it already exists)
if ( !groupList.hasGroup(groupName) ) {
    groupList.addGroup(groupName);
}

// add user to group (unless it is already a member)
XhiveUserIf user = userList.getUser(userName);
XhiveGroupIf group = groupList.getGroup(groupName);
if ( !group.isMember(user) ) {
    user.addGroup(group);
}
```

Samples

[ManageUsers.java](#)

API documentation

[com.xhive.core.interfaces.XhiveAuthorityIf](#)

[com.xhive.core.interfaces.XhiveGroupIf](#)

[com.xhive.core.interfaces.XhiveGroupListIf](#)

[com.xhive.core.interfaces.XhiveUserIf](#)

[com.xhive.core.interfaces.XhiveUserListIf](#)

Using a RAM segment for temporary data

A RAM segment is a special type of database segment that is kept in the database cache but never written to a file. A RAM segment for temporary data can be enabled using the **setTemporaryDataSegment()** method:

```
XhiveDatabaseIf database = session.getDatabase();
database.setTemporaryDataSegment(XhiveDatabaseIf.RAM_SEGMENT_NAME);
```

The xDB dedicated page server program

The dedicated page server is a small Java program that accesses the xDB API. Its purpose is described in the section about [the xDB dedicated page server, page 70](#).

The server process essentially consists of the following code:

```
/*Get a driver for a bootstrap-location*/
XhiveDriverIf driver = XhiveDriverFactory.getDriver(bootstrapPath);
driver.init(cachePages); // Initialize the cache of the driver

ServerSocket socket = new ServerSocket(port); // Create a listen socket
driver.startListenerThread(socket); // Start accepting remote connections
wait(); // Wait forever in this main thread
```

It is important to realize that running xDB with a separate dedicated page server such as configured after installation usually is not the best configuration for performance. The xDB installer configures xDB with a separate server because this is generally a more convenient way to get started, but for production applications where performance is essential it may not be the best choice. Alternatively, Java applications can access the database directly through the bootstrap file, without making any network connection. However, while one process is accessing database files directly, no other processes (including the dedicated server) can *directly* access the database. This need not be a problem: if the direct access application includes the lines related to `startListenerThread` method from the above code segment, it can start a listener thread to accept remote connections from other applications.

Using the FederationSet API

A federation set can run multiple federations in a single page server, using a single page cache and TCP port. For more information, refer to [Federation sets, page 281](#).

The `com.xhive.federationset.interfaces` package provides methods to manipulate a federation set description file, get `XhiveDriverIf` objects, and start federation servers.

Creating a federation set

A federation set can be created like this:

```
String filename = "/path/to/fedsetDescrFile";
XhiveFederationSetFactory.createFederationSet(filename);
XhiveFederationSetIf fs =
    XhiveFederationSetFactory.getFederationSet(filename, null);
Map<String, String> federations = fs.getFederationMap();
federations.put("fdname", "/path/to/bootstrapfile");
```

This would result in a federation set description file like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:federation-set xmlns:ns2="http://www.xhive.com/federationset/schema">
  <federation-list>
    <federation file="/path/to/bootstrapfile" name="fdname"/>
  </federation-list>
```

```
</ns2:federation-set>
```

Using a federation set

You can get a driver to the federation denoted by "fdname" ("/path/to/bootstrapfile") through the federation set server like this:

```
XhivePageCacheIf pageCache =
    XhiveDriverFactory.getFederationFactory().createPageCache(4096);
XhiveFederationSetIf fs =
    XhiveFederationSetFactory.getFederationSet("xhive://hostname:1235/",
        pageCache);
XhiveDriverIf driver = fs.getFederation("fdname");
// Use driver normally, e.g., create session on it
```

To use a federation set internally, the server does not require a start. Only the URI of the federation set is changed in the example code below:

```
XhivePageCacheIf pageCache =
    XhiveDriverFactory.getFederationFactory().createPageCache(4096);
XhiveFederationSetIf fs =
    XhiveFederationSetFactory.getFederationSet("/path/to/federationset",
        pageCache);
XhiveDriverIf driver = fs.getFederation("fdname");
// Use driver normally, e.g., create session on it
```

All federations in the federation set share the single page cache.

There is a convenience method as well:

```
XhiveFederationSetFactory.getFederation()
```

To get a driver to a federation, whether it is accessed through a federation set or not:

```
int numCachePages = ...;
XhiveFederationFactoryIf ff = XhiveDriverFactory.getFederationFactory();
XhivePageCacheIf pc = ff.createPageCache(numCachePages);
XhiveDriverIf driver = XhiveFederationSetFactory.getFederation(path, pc);
```

With `path = "/path/to/fedsetDescrFile#fdname"`, you would get a driver to the federation "fdname" in federation set "/path/to/fedsetDescrFile".

After running a federation set server at port 1235, you would get the same using as path:

```
"xhive://hostname:1235#fdname".
```

The part of the path before the hash (#) is interpreted as a path to a federation set (description file or server). The part of the path after the hash is treated as a path to a federation within the federationset.

Without a hash, method **XhiveFederationSetFactory.getFederation()** interprets the path just like method **XhiveDriverFactory.getDriver()**.

Path `"/path/to/bootstrapfile"` is interpreted as a path to a federation bootstrap file, and path `"xhive://hostname:1235"` as a path to a federation server.

Using xDB with Maven 2

The build tool Maven 2 allows applications to declare dependencies on other projects, and provides automatic resolution of those dependencies and many other useful features. For use with Maven 2, the xDB installation includes a sample POM file.

Prerequisites:

- xDB installed on the local computer
- [Maven 2](#) installed and available on the command line
- Appropriate settings in `$XHIVE_HOME/lib/pom.xml`
- You should be familiar with using the command line and with Maven 2

To set up xDB in your local Maven repository, follow the steps below.

1. Open a command line window in the xDB installation folder.
2. Run the command **`mvn install:install-file -DpomFile=lib\pom.xml -Dfile=lib\xhive.jar`**.

Note: The instructions in this and the following steps assume a Windows command line. If you are on UNIX, replace the backslashes (`\`) with forward slashes (`/`).

This installs xDB into the local Maven repository.

3. Install xDB's Javadoc into the Maven repository.

Note: This step is optional, but provides a better development experience when using an IDE.

- a. Type **`cd docs\apidocs`** to switch to the `docs\apidocs` directory.
- b. Create the *jar* file in the root directory by running **`jar cf ..\..\xhive-javadocs.jar *`**
- c. Type **`cd ..\..`** to move back to the top folder of the xDB installation.
- d. Type **`mvn install:install-file -DpomFile=lib\pom.xml -Dfile=xhive-javadocs.jar -Dclassifier=javadoc`** to install the Javadoc jar into the Maven repository.
- e. Afterwards, you can delete the file `xhive-javadocs.jar` from your xDB directory.

You can declare a dependency on xDB in Maven-based projects by including the following in your POM's dependency section (make sure to set `<version/>` to your current xDB version):

```
<dependency>
  <groupId>com.xhive</groupId>
  <artifactId>xdb</artifactId>
  <version>10.5</version>
</dependency>
```

Using xDB with Spring

xDB supports convenient declarative transactions when used in combination with the [Spring framework](#). Applications can configure an xDB transaction manager in their Spring application context and use Spring's transaction management facilities to manage xDB transactions. This requires general

knowledge of the Spring framework, and in particular Spring data sources. This text does **not** discuss xDB's XA transaction support, which can also be used in conjunction with Spring.

xDB provides three classes to support Spring transactions:

- **XhiveDataSource** is comparable to a JDBC session pool. This class holds the XhiveDriverIf reference and pools XhiveSessionIf objects for your application. First read connection settings from a properties file, connection.properties in the WEB-INF folder of your war file, and then pass the bootstrap, database, username, and password values for XhiveDataSource's constructor.
- **XhiveTransactionManager** is a Spring TransactionManager that will handle transactions according to Spring's rules. In particular, it will open transactions, make sure to commit them or roll them back after web requests, and handle configuration such as the readOnly = true flag in the example below.
- **XhiveSessionAccess** provides access to the xDB transaction. You need to explicitly tell Spring about its presence, so that Spring can automatically inject it into your code.

Note: The Maven 2 POM file included in the xDB distribution (\$XHIVE_HOME/lib/pom.xml) specifies all necessary dependencies for developing Spring-based xDB applications. See [Using xDB with Maven 2, page 94](#) for more information.

The following example shows a Spring application context configuration, using these classes to provide transactional support.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.2.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/context/spring-tx-3.2.xsd">
  <context:property-placeholder location="/WEB-INF/connection.properties"/>
  <context:annotation-config/>
  <tx:annotation-driven/>
  <bean class="com.xhive.spring.XhiveSessionAccess"/>
  <bean id="transactionManager" class="com.xhive.spring.XhiveTransactionManager"/>
  <bean id="dataSource" class="com.xhive.spring.XhiveDataSource">
    <constructor-arg value="${sessionPool.bootstrap}"/>
    <constructor-arg value="${sessionPool.database}"/>
    <constructor-arg value="${sessionPool.username}"/>
    <constructor-arg value="${sessionPool.password}"/>
  </bean>
</beans>
```

After configuring the transactional support, it can be used in Java code like this:

```
@Controller
public class TestController {
    @Autowired
    private XhiveSessionAccess acc;

    @Transactional(readOnly = true)
    @RequestMapping("/test-{name}.html")
    public ModelAndView test(@PathVariable String name) {
```

```
HashMap<String, Object> params = Maps.newHashMap();
params.put("name", name);
XhiveXQueryValueIf result = acc.query("document { <result>Hello, { $name }," +
    " it is now { current-dateTime() }</result> }", params).next();
return new ModelAndView("main", "xmlSource", new DOMSource(result.asNode()));
}
}
```

API documentation

[XhiveDataSource](#)

[XhiveTransactionManager](#)

[XhiveSessionAccess](#)

xDB and OSGi

OSGi (Open Service Gateway initiative) is a framework for developing and deploying modular software programs and libraries in Java. OSGi applications are built of *bundles* which are dynamically loadable collections of Java classes, jars, and configuration files that explicitly declare their external dependencies.

xDB supports deployment in an OSGi environment in two ways. The first approach is to simply use the main library `$XHIVE_HOME/lib/xhive.jar`. Because this library is at the same time an OSGi bundle, it can be deployed in an OSGi container easily.

The second approach is to use the libraries `$XHIVE_HOME/lib/osgi/xhive-api.jar` and `$XHIVE_HOME/lib/xhive-impl.jar`. These libraries are OSGi bundles that decouple the xDB interfaces (`xhive-api.jar`) from their actual implementation (`xhive-impl.jar`). The implementation bundle uses OSGi [Declarative Services](#) to register itself as the implementation of the following API services:

- **com.xhive.core.interfaces.XhiveDriverFactoryIf** - a service for obtaining xDB **XhiveDriverIf** implementations; and
- **com.xhive.federationset.interfaces.XhiveFederationSetFactoryIf** - a service for creating and retrieving xDB federation sets.

Depending on the requirements, one of the above deployment options may be preferable over the other.

Note: The third-party libraries that are included in the xDB distribution are *not* OSGi-compatible. In order to use xDB with OSGi, OSGi-enabled versions of these libraries must be obtained and deployed in the OSGi container. The Ant build script `$XHIVE_HOME/bin/build.xml` contains a simple functionality for converting the core xDB dependencies into OSGi bundles - however, its use for anything beyond executing the xDB sample applications is discouraged. Refer to the *Import-Package* OSGi manifest headers in `xhive.jar` (or `xhive-api.jar` and `xhive-impl.jar`, respectively) for the exact packages (and versions of thereof) that the xDB bundles depend on.

The xDB distribution comes with a simple OSGi sample application. The application source code and configuration files can be found in the directories `src/samples/osgi/` and `src/samples/etc/osgi/`, respectively. Follow the instructions in `$XHIVE_HOME/bin/build.xml` on how to build and deploy the sample application.

Using xDB with JAAS

xDB provides user authentication, but also allows utilizing external authentication systems based on Java Authentication and Authorization Service (JAAS), such as authentication for LDAP databases or operating systems. JAAS offers a plug-in authentication module framework with modules for different types of authentication systems.

JAAS handles the user authentication in combination with xDB. Once JAAS authentication is successful, xDB automatically creates users and groups within xDB that match the authentication information.

Using JAAS authentication with xDB requires implementing certain Java interfaces and configuring the Pluggable Authentication Module (PAM), as follows:

- Specifying the PAM server connection in the JAAS configuration file or Java code.
- Enabling JAAS on the xDB driver object using the **`driver.getSecurityConfig().enableJavaAuthentication(chosenConfigurationEntryName, XhiveNameHandlerIf)`** method. **Note:** This method can only be called for drivers that connect directly to the bootstrap file. The standard dedicated server included with xDB cannot be configured for JAAS and cannot be used in combination with JAAS.
- Providing a custom implementation for the **`XhiveNameHandlerIf`** argument to map JAAS user/group objects to xDB user names and group names.
- Connecting a session using the **`connect(databaseName, CallbackHandler)`** method.

The callback handler is a JAAS interface that allows passing authentication parameters to the PAM server. Users can provide their own implementation or use standard classes, such as the **`DialogCallbackHandler`** class that the administration client uses for JAAS authentication.

The LDAP sample code included with xDB shows the interfaces that must be implemented and the API calls that enable JAAS authentication.

Samples

[../ldap/SampleClient.java](#)

[../ldap/XhiveServerWithLDAP.java](#)

API documentation

[com.xhive.core.interfaces.XhiveDriverIf](#)

[com.xhive.core.interfaces.XhiveSessionIf](#)

Using the API with SSL

Configuring the server for SSL

When starting an xDB server through the **`XhiveDriverIf.startListenerThread()`** API, you must pass a **`java.net.ServerSocket`** to listen to. To use SSL, pass a **`javax.net.ssl.SSLServerSocket`** object that has

been created with any required options. The API can also be used for more complex features, like using your own key manager, specifying cipher suites, and other more specialized SSL configuration.

Configuring the client for SSL

Configuring SSL on the client includes specifying a URL of the format `xhives://host:port` as the argument in the **XhiveDriverFactory.getDriver()** method.

To do anything special, like using a custom trust manager, pass the same URL to the **XhiveDriverFactory.getDriver()** method, and also pass a `SocketFactory` object to the **XhiveDriverIf.init(int cachePages, SocketFactory socketFactory)** call.

Managing Documents in Applications

This chapter contains the following topics:

- **Creating and managing documents**
- **Parsing XML documents**
- **Validating XML documents**
- **Storing XML documents**
- **DOM configuration settings**
- **Importing non-XML data**
- **Creating a document**
- **Retrieving documents and document parts**
- **Traversing XML documents**
- **Exporting XML documents**
- **Publishing XML documents**
- **XLink interfaces**
- **Using versioning**
- **Working with versioned documents**
- **Using searchable versions**
- **Using metadata on library children**
- **Using abstract schemas**
- **Using the APIs for XSL transformations**

Creating and managing documents

xDB stores XML data as documents. A document is represented in the xDB API using the **org.w3c.dom.Document** interface. This interface includes methods for creating an XML document, updating XML documents, and for accessing document parts, such as elements, comments, and attributes.

Parsing XML documents

To import an XML document from an external source, the document must be parsed. xDB supports the DOM Load and Save specification, which provides standard ways for parsing and serializing

DOMs. For more information about the specification, see the [W3C Document Object Model \(DOM\) Level 3 Load and Save Specification](#).

The **XhiveLibraryIf** interface extends the **DOMImplementationLS** interface, which can be used to create **LSParser** and **LSSerializer** objects. You can use the **parseURI** method of the **LSParser** interface for parsing documents. LSParser objects must be created on the library where the parsed documents are stored.

Example

The example below uses the `parseURI` method of the DOM Load/Save **LSParser** interface to parse a document. When the document is parsed successfully, a DOM `Document` object is returned.

```
LSParser builder = rootLibrary.createLSParser();
Document firstDocument = builder.parseURI( new File(fileName).toURL().toString());
```

Note: An explicit **appendChild** is required to store a parsed document in the database, otherwise the parsed document not stored.

Related topics

[Storing XML documents](#)

[Validating XML documents](#)

Samples

[ParseDocuments.java](#)

[DOMLoadSave.java](#)

API documentation

[org.w3c.dom.as](#)

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

[org.w3c.dom.ls.LSParser](#)

[org.w3c.dom.ls.LSSerializer](#)

Parse with context

Complete documents can be parsed with context using the **parseWithContext** function on the **LSParser** method, as in the following example:

```
LSParser parser = charterLib.createLSParser();
// Using (null, null, null) as arguments means the document will be completely empty
Document document = charterLib.createDocument(null, null, null);
// Other actions on document...
LSInput source = charterLib.createLSInput();
source.setSystemId("file:///c:/docs/document.xml");
parser.parseWithContext(source, document, LSParser.ACTION_REPLACE);
```

This involves more code than a simple parse, but has the advantage that you can first create the document, then set indexes on it (at the 'Other actions on document...' line), and then parse the data on it. Indexing documents during parsing may bring some performance gains, especially for large documents.

Note:

Samples

[ParseDocumentsWithContext.java](#)

API documentation

[org.w3c.dom.ls.LSParser](#)

[org.w3c.dom.ls.LSSerializer](#)

[com.xhive.dom.interfaces.XhiveNodeIf](#)

Validating XML documents

The XML validation process validates an XML document and stores the DTD or XML schema information as an abstract schema model (ASModel) in the catalog.

When a document is validated, xDB checks whether the catalog contains an abstract schema that matches either the DTD or the XML schema. If so, validation uses the abstract schema instead of the external DTD. Catalog checking can be turned off by setting the **xhive-ignore-catalog** parameter **LSParser** object to **true**. When catalog checking is turned off, the validation process always creates a new abstract schema for each validated document.

Note: If a system ID is used to refer to a DTD, the DTD is stored for every document that is parsed. Documents using only a system ID can be parsed without storing the schema by setting the **xhive-ignore-catalog** parameter to **false**.

Examples

By default the **validate** configuration parameter is disabled and must be enabled to parse a file with validation. The following example describes how to enable parsing with validation.

```
LSParser parser = charterLib.createLSParser();
parser.getDomConfig().setParameter("validate", Boolean.TRUE);
Document firstDocument = parser.parseURI( new File(fileName).toURL().toString());
```

If the parsed document contains a reference to a DTD, the DTD is stored as an ASModel within the library catalog.

The **XhiveCatalogIf** interface in the **com.xhive.dom.interfaces** package contains several methods for updating and querying abstract schema models. The following example retrieves a catalog and the abstract schema from the root library.

```
// retrieve the catalog of the "UN Charter" library
XhiveCatalogIf unCharterCatalog = charterLib.getCatalog();

// get the abstract schema models that exist in the root library catalog
// do not include models from locations higher in the tree
Iterator<ASModel> iter = unCharterCatalog.getASModels(false);
while (iter.hasNext()) {
    ASModel asModel = iter.next();
    System.out.println(" asModel = " + asModel.getLocation());
}
```

Samples

[ParseDocumentsWithValidation.java](#)

[ParseDocumentsWithContext.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

[org.w3c.dom.ls.LSParser](#)

Normalizing XML documents

xDB includes the **normalizeDocument** function for normalizing XML documents in the **XhiveDocumentIf** interface. The complete list of normalization options can be found in the Javadoc documentation for the **DOMConfiguration** interface.

Examples

The following code example describes how to normalize a document using the **DOMConfiguration** interface.

```
DOMConfiguration config = ((XhiveDocumentIf) document).getDomConfig();
config.setParameter("validate", Boolean.TRUE);
config.setParameter("error-handler", new SimpleDOMErrorPrinter());
document.normalizeDocument();
```

Normalizing a document with validation requires enabling the "validate" parameter. The **normalizeDocument** method does not throw any exceptions. An error handler can be set during normalization.

The "schema-location" parameter can be used to validate against a different schema. Setting a schema location also requires setting the schema type. The following code example describes how to set a schema-location.

```
config.setParameter("schema-type", "http://www.w3.org/2001/XMLSchema");
config.setParameter("schema-location", "personal.xsd");
```

If the schema-location is set, the validation process first searches for a corresponding XML schema in the catalog. If no schema is found, the validation process searches for a schema in the file system. During document parsing, the schema-location is resolved relative to the document URI. The document URI is not available during validation. A full path must be set if a document is validated against a schema located in the file system.

Setting PSVI information

If the **xhive-psvi** parameter in the **LSParser** object is enabled, PSVI information can be set when a document is parsed or normalized. If a document is parsed without validation, PSVI information can be set during validation.

Related topics

[Post Validation Schema Infoset \(PSVI\)](#)

[Accessing PSVI information](#)

Samples

[ValidateDocumentWithXMLSchema.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveDocumentIf](#)

[org.w3c.dom.DOMConfiguration](#)

Storing XML documents

The **appendChild()** method is used to store XML documents in an xDB database. This method requires a handle to the library where the document is going to be stored.

The following example shows how to store a document in the root library of the sample database:

```
XhiveLibraryIf rootLibrary = united_nations_db.getRoot();
rootLibrary.appendChild(firstDocument);
```

Alternatively, documents can be stored using the **insertBefore()** method, which is also a standard DOM method. In the following example, `firstDocument` specifies the name of the new document and `secondDocument` specifies the name of the existing document in front of which the new document is stored.

```
rootLibrary.insertBefore(secondDocument, firstDocument);
```

Where

Samples

[StoreDocuments.java](#)

API documentation

[org.w3c.dom.Node](#)

DOM configuration settings

The DOM level 3 **DOMConfiguration** interface is used to set Boolean, string, and user object parameters. The Boolean configuration settings and string parameters of a document are stored in the database. The **normalizeDocument** method of the **XhiveDocumentIf** interface uses these settings. The **XhiveDocumentIf**, **LSParser** and **LSSerializer** objects each have their own configuration object.

The options are listed in the JavaDOC documentation of the **DOMConfiguration** interface, and the **getDomConfig()** method of the **LSParser** and **LSSerializer** objects.

The following code example shows how to set a Boolean parameter and user object parameter:

```
LSParser parser = library.createLSParser();
XhiveDocumentIf document =
    (XhiveDocumentIf) parser.parseURI(new File(fileName).toURL().toString());
DOMConfiguration config = document.getDomConfig();
config.setParameter("validate", Boolean.TRUE);
```

```
config.setParameter("error-handler", new SimpleDOMErrorPrinter());
```

The xDB default configuration settings conform to the default settings defined by the [Document Object Model \(DOM\) Level 3 Core Specification](#) and the [Document Object Model \(DOM\) Level 3 Load and Save Specification](#). A supported parameter can be set to another value.

The following code example shows how to test whether a configuration supports a Boolean parameter value:

```
config.canSetParameter("validate", Boolean.TRUE);
```

Deviations from specification

In xDB, the **LSParser** default value is set to **"Boolean.FALSE"**. According to the **LSParser** object specification, the default value for the **"element-content-whitespace"** Boolean parameter is **"Boolean.TRUE"**. Documents that are parsed and stored with this setting can have many text nodes containing only spaces. These additional nodes need more space on the disk and can adversely affect query and validation performance.

Additional parameters

xDB adds some parameters for use by **LSParser** and/or **Document** objects.

Table 13 Additional parameters

Parameter name	Interface	Default value	Description
xhive-ignore-catalog	XhiveDocumentIf, LSParser	Boolean.FALSE	Specifies whether or not to ignore the corresponding DTD's and XML schemas in the catalog during validated parsing.
xhive-store-schema	XhiveDocumentIf, LSParser	Defaults to the value of the configuration setting "validate" (Boolean.TRUE or Boolean.FALSE). Can be overridden after setting "validate".	Specifies whether or not to store the corresponding DTD's or XML schemas in the catalog during validated parsing.
xhive-psvi	LSParser	Boolean.FALSE	Specifies whether to store PSVI information for elements and attributes, and enables access to PSVI information and XQuery data type support.
xhive-store-schema-only-internal-subset	LSParser	Boolean.FALSE	Specifies whether to store only the internal subset of the document and no external subsets. This parameter only applies in conjunction with DTDs.
xhive-predefined-entities	LSParser	Boolean.FALSE	Specifies whether to store predefined entities as entity reference nodes.

Parameter name	Interface	Default value	Description
xhive-character-references	LSParser	Boolean.FALSE	Specifies whether to store character references as entity reference nodes. If this parameter is set to Boolean.TRUE, the xDB creates entity reference nodes names that do not comply with the DOM recommendation.
xhive-raw-attributes	LSParser, LSSerializer	Boolean.FALSE	Specifies whether to store the raw attribute value when parsing or serializing a document.
xhive-insert-xmlbase	LSParser, LSSerializer	Boolean.FALSE	<p>When parsing a document, this parameter specifies whether to set the xml:base attribute for the top level element read from an external parsed entity. Setting the xml:base attribute ensures that the Node.getBaseURI() method returns correct results.</p> <p>When serializing a document, this parameter specifies whether to insert xml:base attributes when external parsed entities are expanded. This parameter only applies when the "entities" option is set to false.</p>
xhive-sync-features	LSParser	Boolean.FALSE	Specifies whether the parameter values of the XhiveDocumentIf interface are synchronized with the LSParser interface parameter values. The "xhive-psvi" and "schema-location" values are always synchronized.
xhive-schema-ids	Read-only parameter of the XhiveDocumentIf interface.	null	Specifies the identification of XML schema ids used by the document.
xhive-node-callback	LSParser, XhiveDocumentIf	null	<p>Enables applications to call a user-defined instance of the XhiveNodeCallbackIf interface before importing or constructing text or CDATASection nodes. The function specifies whether the nodes should be compressed.</p> <p>Compression should not be used for all text or CDATASection nodes because the compressed text representation header adds additional overhead and the compression algorithms consumes CPU time. xDB stores text or the</p>

Parameter name	Interface	Default value	Description
			CDATASection in a compressed representation only if the compressed text size is smaller than the original text size.
xhive-security-manager	LSParser	null	Sets an instance of the org.apache.xerces.util.SecurityManager class to be used during parsing. When parsing untrusted XML, this can prevent malicious documents from using excessive resources.

Samples

[DOMLoadSave.java](#)

[TextCompression.java](#)

API documentation

[org.w3c.dom.DOMConfiguration](#)

[org.w3c.dom.ls.LSParser](#)

[org.w3c.dom.ls.LSSerializer](#)

[com.xhive.dom.interfaces.XhiveDocumentIf](#)

Importing non-XML data

xDB can import data from non-XML files.

- The **com.xhive.util.interfaces.XhiveSQLDataLoaderIf** can import data from JDBC-compliant relational databases.
- The **com.xhive.util.interfaces.XhiveCSVFileLoaderIf** interface can import data in CSV format.

For more information, see the API documentation.

Note: The **XhiveSqlLoaderIf** interface is deprecated. Use the **XhiveSQLDataLoaderIf** and **XhiveCSVFileLoaderIf** interfaces instead.

CSV Import Example

The example code below uses **XhiveCSVFileLoaderIf** to import data in CSV format into xDB, and stores the data as an XML document.

```
// get the XhiveCSVFileLoader
FileInputStream input = new FileInputStream(fileName);
XhiveCSVFileLoaderIf loader = XhiveDriverFactory.getDriver().getCSVFileLoader(input);

// Set the options to load the data with
loader.setSeparator(',');
```

```

loader.setEscape('\\');
loader.setEnclose('"');
loader.setHeadersIncl(true);
loader.setRowName("member");
loader.setColumnNames(new String[]{"name", "admission_date", "additional_note"});
loader.setColumn2Attribute(new boolean[] { false, false, false });

// Create a document that will serve as target for the data
Document unMembersDoc = rootLibrary.createDocument(null, "UN_members", null);

// Set document(-element) as the target for the load
loader.setTargetDocument(unMembersDoc, unMembersDoc.getDocumentElement());

// Perform the actual load
loader.loadCSVData();

```

From the following CSV data:

```

"Member", "Date of Admission", "Additional Notes"
"Iceland",19 Nov. 1946, ""
"India",30 Oct. 1945, ""
"Indonesia",28 Sep. 1950, "By letter of 20 January ..."
"Iran (Islamic Republic of)",24 Oct. 1945, ""

```

the example code will generate the following XML document:

```

<UN_members>
<member>
<name>Iceland</name>
<admission_date>19 Nov. 1946</admission_date>
<additional_note></additional_note>
</member>
<member>
<name>India</name>
<admission_date>30 Oct. 1945</admission_date>
<additional_note></additional_note>
</member>
<member>
<name>Indonesia</name>
<admission_date>28 Sep. 1950</admission_date>
<additional_note>By letter of 20 January 1965...</additional_note>
</member>
<member>
<name>Iran (Islamic Republic of)</name>
<admission_date>24 Oct. 1945</admission_date>
<additional_note></additional_note>
</member>
</UN_members>

```

Samples

[StoreRelationalData.java](#)

API documentation

[com.xhive.util.interfaces.XhiveSQLDataLoaderIf](#)

[com.xhive.util.interfaces.XhiveCSVFileLoaderIf](#)

Creating a document

To create a document:

1. Obtain a handle to a DOM implementation with `rootLibrary` because **XhiveLibraryIf** extends `DOMImplementation`, as follows:

```
DOMImplementation impl = rootLibrary;
```

2. Create a `DocumentType` object and a `Document` object using the `createDocument()` method in `org.w3c.dom.DOMImplementation`, as follows:

```
DocumentType docType = impl.createDocumentType("typeName", "publicId", "systemId");  
Document eventsDocument = impl.createDocument(null, "events", docType);
```

Because no namespaceURI is used, the first parameter can be left empty. The second parameter of the `createDocument()` method, `events`, is the tag name of the root element. The third parameter sets the `docType` of the new document.

3. Obtain a handle to the root element of the newly created document, as follows:

```
Element rootElement = eventsDocument.getDocumentElement();
```

4. Add document parts using standard DOM methods.

These methods are located in the `org.w3c.dom.Document` interface. The most commonly used methods are:

- `createAttribute()`
- `createComment()`
- `createElement()`
- `createTextNode()`

Examples

The following code adds a comment, an element named `event` with attribute occurrence, and the text value "UNICEF, Executive Board, annual session" to the new document:

```
// add a comment to the document before the root element  
Comment comment = eventsDocument.createComment("this document contains UN events");  
eventsDocument.insertBefore(comment, rootElement);  
  
// add a new element to root element  
Element eventElement = eventsDocument.createElement("event");  
rootElement.appendChild(eventElement);  
  
// add text value to the element  
Text eventText =  
    eventsDocument.createTextNode("UNICEF, Executive Board, annual session");
```

```

eventElement.appendChild(eventText);

// add an attribute to the element
eventElement.setAttribute("occurrence", "year");

Add date element with the value "4-8 June, 2001" to the event element, as follows:

// add a new element to event
Element dateElement = eventsDocument.createElement("date");
eventElement.appendChild( dateElement );

// add text value to the date element
Text dateText = eventsDocument.createTextNode("4-8 June, 2001");
dateElement.appendChild(dateText);

```

The resulting XML document looks as follows:

```

<!DOCTYPE typeName PUBLIC "publicId" "systemId">
<!--this document contains UN events-->
<events>
  <event occurrence="year">
    UNICEF, Executive Board, annual session
    <date>4-8 June, 2001</date>
  </event>
</events>

```

Use the `appendChild()` or `insertBefore()` method to store the document in the database. The following code uses `appendChild()` to store the document in the root library:

```
rootLibrary.appendChild(eventsDocument);
```

Samples

[CreateDocument.java](#)

API documentation

[org.w3c.dom.Document](#)

[org.w3c.dom.DOMImplementation](#)

Retrieving documents and document parts

xDB offers various ways to retrieve documents and document parts from the database. For example, documents can be retrieved using DOM operations, document ID, document name, XQuery, indexing, XPointer and XPath.

Samples

[RetrieveDocuments.java](#)

[RetrieveDocumentParts.java](#)

API documentation

[org.w3c.dom.Node](#)

[org.w3c.dom.Element](#)

[com.xhive.dom.interfaces.XhiveLibraryChildf](#)

[com.xhive.dom.interfaces.XhiveLibrarylf](#)

[com.xhive.dom.interfaces.XhiveNodeIf](#)

Using DOM operations

The DOM specifications include some methods for retrieving documents or document parts. These methods are part of the **org.w3c.dom.Node** interface. Elements within the DOM are linked as parent-child, or siblings.

Some DOM operations are described in [DOM operations for retrieving documents, page 110](#).

Table 14 DOM operations for retrieving documents

Interface	Methods	Description
org.w3c.dom.Node	getFirstChild(), getLastChild(), getPreviousSibling(), getChildNodes() getNextSibling(), hasChildNodes()	Methods for retrieving document children. Methods for retrieving specific document parts.
org.w3c.dom.Element	getAttribute(), getAttributeNode()	Methods for retrieving element attributes within a document.
com.xhive.dom.interfaces.XhiveNodeIf	getFirstChildByType(), getFirstChildElementByName()	Methods for retrieving document children by type and name. The interface extends the functionality of the org.w3c.dom.Node interface.

Note: Retrieving all children of a node using the `getChildNodes()` method can be slow. The `getNextSibling()` method is a faster way to iterate across child nodes

Examples

The following example code checks whether a library has any children and counts the number of children.

```
int nrChildren = 0;
Node n = charterLib.getFirstChild();
while(n != null) {
    nrChildren++;
    n = n.getNextSibling();
}
```

The following example code displays all elements within an XML document.

```

public static void showChildren (Node theNode, int level) {

    // some output formatting
    String indentation = "";
    for (int i=0; i<level; i++) {
        indentation += "\t";
    }

    Node n = theNode.getFirstChild();

    int j = 1;

    // as long as there are children...
    while(n != null) {

        // and child is of type 'element'...
        if ( n.getNodeType() == org.w3c.dom.Node.ELEMENT_NODE ) {
            // show the element...
            System.out.println(indentation + "child " + j++ + " is a " + n.getNodeName());

            // and get the children of this element (recursively)
            showChildren(n, level+1);
        }

        // get next child
        n = n.getNextSibling();
    }
}

```

Using document ID

xDB automatically assigns an identifier to a new document. This identifier is unique within the context of a library. The **get()** method in the **com.xhive.dom.interfaces.XhiveLibraryIf** interface retrieves documents by identifier.

Example

The following code example retrieves a document by identifier using the `get()` method.

```

int anId = 10;
Node child = charterLib.get(anId);
System.out.println("document with ID = " + anId + " in \"UN Charter\"
    has name: " + ((XhiveLibraryChildIf)child).getName());

```

Using document name

Although every document has an identifier, it can be more convenient to retrieve documents by their name. Document names are optional but must be unique within the context of the library in which the document is stored. The **get()** method in the **com.xhive.dom.interfaces.XhiveLibraryIf** interface retrieves documents by name.

Example

The following code example retrieves a document by name using the `get()` method.

```
String documentName = "UN Charter - Chapter 2";
Document docRetrievedByName = (Document) charterLib.get( documentName );
```

Using XQuery

XQueries can be run on libraries and documents using the **executeXQuery(String query)** method in the **XhiveLibraryChildIf** interface. The method returns a result sequence and each result element is an instance of the **XhiveXQueryValueIf** object.

Example

The following example code executes a query that retrieves all chapter titles of a document.

```
Iterator result = charterLib.executeXQuery("//chapter/title");
while (result.hasNext()) {
    XhiveXQueryValueIf value = (XhiveXQueryValueIf) result.next();
    // We know this query will only return nodes.
    Node node = value.asNode();
    // Do something with the node ...
}
```

Samples

[XQuery.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveLibraryChildIf](#)

[com.xhive.query.interfaces](#)

Using indexes

Using indexes can dramatically improve query performance. For more information about indexes, see [Indexes](#), page 150.

Using XPointer with library path

The **executeFullPathXPointerQuery()** method in the **com.xhive.dom.interfaces.XhiveLibraryChildIf** interface retrieves documents by library path and document name.

Examples

The following example code retrieves and displays the document titled UN Charter - Chapter 2 from the UN Charter library.

```
Iterator docsFound = rootLibrary.executeFullPathXPointerQuery(
    "/UN Charter/UN Charter - Chapter 2");
Document docRetrievedByFPXPQ = (Document) docsFound.next();
System.out.println(docRetrievedByFPXPQ.toString());
```


The following example code specifies a relative path to retrieve the document.

```
// newLib is a sub library of "UN Charter"

// execute the FullPathXPathQuery relative to the new sub library
docsFound = newLib.executeFullPathXPathQuery("../UN Charter - Chapter 3");
docRetrievedByFPXPQ = (Document)docsFound.next();
System.out.println(docRetrievedByFPXPQ.toString());
```

When the document name is not specified, the `executeFullPathXPathQuery()` method returns a library, as described in the following example.

```
Iterator librariesFound = rootLibrary.executeFullPathXPathQuery("/UN Charter");
XhiveLibraryIf charterLib = (XhiveLibraryIf)librariesFound.next();
```

Using an XPath expression

If the library path contains an XPath expression, the expression can be used to retrieve parts of a document.

The input query uses the following syntax:

```
/libname[/libname...][/docname|/id:docid[##versionId]][#xpath_query]
```

Optional parameters are enclosed in [].

Adding an XPath expression to the query string allows using the `executeFullPathXPathQuery()` method to retrieve parts of an XML document.

Examples

The following example retrieves all `title` elements from the `Un Charter - Chapter 5` sample document in the `UN Charter` library.

```
String sampleLibName = "/UN Charter";
String sampleDocName = "UN Charter - Chapter 5";
String sampleDocPath = sampleLibName + "/" + sampleDocName;
String queryXPath = "#xpath(/descendant::title)";

Iterator resultNodes = rootLibrary.executeFullPathXPathQuery(
    sampleDocPath + queryXPath);
while ( resultNodes.hasNext() ) {
    Node resultNode = (Node)resultNodes.next();
    System.out.println( resultNode.getFirstChild().getNodeValue() );
}
```

The following example code retrieves the first paragraph of UN article #68 without specifying which document contains this article.

```
queryXPath = "#xpath(/descendant::article[@number='68']/para[1])";

// note that we only specify the library path and not the document name:
resultNodes = rootLibrary.executeFullPathXPathQuery(sampleLibName + queryXPath);
while ( resultNodes.hasNext() ) {
    Node resultNode = (Node)resultNodes.next();
    System.out.println( resultNode.getFirstChild().getNodeValue() );
}
```

The `executeFullPathXPathQuery()` method can retrieve a specific document version by adding `##` followed by a version ID or label after the document name. The method first evaluates the version identifier as a label. If a version with that label cannot be found, the version identifier is treated as a version ID.

The following code example retrieves all `title` elements for version 1.3 of the UN Charter - Chapter 5 sample document in the UN Charter library .

```
String sampleLibName = "/UN Charter";
String sampleDocName = "UN Charter - Chapter 5";
String sampleDocPath = sampleLibName + "/" + sampleDocName;
String versionIdentifier = "##1.3";
String queryXPather = "#xpather(/descendant::title)";

Iterator resultNodes = rootLibrary.executeFullPathXPathQuery(
    sampleDocPath + versionIdentifier + queryXPather);
while ( resultNodes.hasNext() ) {
    Node resultNode = (Node)resultNodes.next();
    System.out.println( resultNode.getFirstChild().getNodeValue() );
}
```

For more information about XPath queries, see [Using XPath and XPather, page 114](#).

Using XPath and XPather

Note: The information in this topic is mostly for backward compatibility. Please use XQuery whenever possible.

Using XPath

For XPath queries, you can use the `executeXPathQuery(...)` methods in the `XhiveNodeIf` interface. Optionally, you can supply a query context that contains namespace declarations, variable and function bindings, and an absolute root.

The results of an XPath query are represented by the `XhiveQueryResultIf` interface. For example, the following code executes a query that retrieves all chapter titles of a document.

```
XhiveQueryResultIf result = charterLib.executeXPathQuery("descendant::chapter/title");
```

The `XhiveQueryResultIf` interface includes methods for extracting different types of information from a query result. The result can be of one of several types: a string, a Boolean, a number or a location set. A location set is a collection of nodes, points, and ranges. For information about the rules that determine the outcome of conversion, refer to the XPath specifications.

To convert a query result, you can use the following methods to extract different result types:

- `getStringValue()` - Retrieves the string value of a result.
- `getBooleanValue()` - Retrieves the Boolean value of a result.
- `getNumberValue()` - Retrieves the numeric value of a result.
- `getLocationSetValue()` - Retrieves the location set value of a result.

The following code processes the results returned as a location set.

```
if (result != null){
    if ( resultType == XhiveQueryResultIf.LOCATIONSET ){
```

```

XhiveLocationIteratorIf resultNodeSet = result.getLocationSetValue();
XhiveLocationIf resultNode;

while ( resultNodeSet.hasNext() ) {
    resultNode = resultNodeSet.next();
    if ( resultNode.getLocationType() == Node.ELEMENT_NODE ) {
System.out.println(" " + ((Node)resultNode).getFirstChild().getNodeValue());
    }
}
}
}
}

```

Note: xDB throws an **XhiveException** if the **getLocationSetValue()** method is called on a result that is not a location set.

Using XPointer

The XPointer XML Pointer Language is based on the XPath XML Path Language. XPointer supports addressing the internal structures of XML documents to traverse a hierarchical document structure and select parts of the hierarchy based on various properties. For a complete and up-to-date description of XPointer, refer to the [XML Pointer Language \(XPointer\) Version 1.0 documentation](#) at the W3C website.

XPointer queries can be executed in a similar way as XPath queries:

```

XhiveQueryResultIf result = charterLib.executeXPointerQuery(
    "xpointer(/chapter/article/para/list/item[1]/para)");

```

The result of an XPointer query has to be a non-empty location set or an exception is thrown.

Working with namespaces

Before a namespace can be used in an XPath query, the namespace must be declared in an **XhiveXPathContextIf** object. The object can be supplied using the **addNamespaceBinding(...)** method when executing the query, as described in the following code example.

```

XhiveXPathContextIf xpathContext = nsDocument.createXPathContext();

// Add a namespace declaration for the xsl-namespace
// Note that the prefix does not have to match
xpathContext.addNamespaceBinding("ns", "http://www.w3.org/1999/XSL/Transform");

// Execute the query
XhiveQueryResultIf result = nsDocument.executeXPathQuery(
    "descendant::ns:template/@match", xpathContext);

```

Note: **XhiveXPathContextIf** objects become invalid after the end of a commit, checkpoint, or rollback transaction to prevent the context operating on invalid data. The view of the database changes after the beginning of a new transaction. If a context is used after a transaction has ended an **XhiveException.INVALID_CONTEXT** exception is thrown.

In *XPointer*, the namespace declaration is included in the query. The format of the declaration is defined in the [XPointer specifications](#):

```

theQuery = "xmlns(ns=http://www.w3.org/1999/XSL/Transform) xpointer(
    descendant::ns:template/@match)";

```

Samples

[XPath.java](#)

[XPathXPathNamespaces.java](#)

API documentation

[com.xhive.query.interfaces](#)

[com.xhive.xpath.interfaces.XhiveXPathContextIf](#)

[com.xhive.dom.interfaces.XhiveNodeIf](#)

Traversing XML documents

Traversal can be used to perform an action, such as applying a change, on some nodes. A traversal moves through a tree processing every node it encounters.

In xDB, you can traverse XML documents using:

- [DOM Traversal](#), page 116
- [Function objects](#), page 119

Samples

[DomTraversal.java](#)

[MyNumberFinder.java](#)

[FunctionObjects.java](#)

API documentation

[org.w3c.dom.traversal](#)

[com.xhive.dom.interfaces.XhiveNodeIf](#)

[com.xhive.dom.interfaces.XhiveFunctionIf](#)

Using DOM traversal

The DOM Level 2 Traversal specification specifies operations for traversing XML documents in the `org.w3c.dom.traversal` package. The package defines two different types of traversal:

- **NodeIterator**

This type traverses a flat representation of an XML document.

- **TreeWalker**

This type traverses a tree representation of an XML document.

For example, the simple XML document

```
<A>
  <B>some text</B>
  <C>
    <D>1st child of C</D>
```

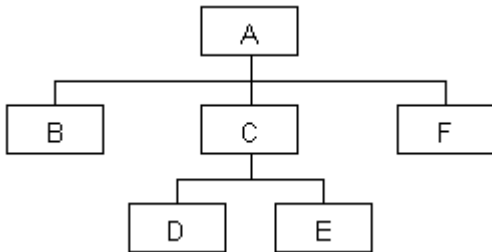
```
<E>2nd child of C</E>
</C>
<F>some more text</F>
</A>
```

is associated with the flat representation

```
A B C D E F
```

and the tree representation.

Figure 4 XML document tree



The interfaces and methods used for traversal are described in the [Traversal interfaces and methods, page 117](#) table.

Table 15 Traversal interfaces and methods

Interface	Methods	Description
NodeIterator	nextNode(), previousNode()	Methods for retrieving the next and previous node in a traversal.
TreeWalker	nextNode(), previousNode(), parentNode(), firstChild(), lastChild(), previousSibling(), nextSibling()	Methods for retrieving the next and previous node or sibling, and first and last child in a traversal.
NodeFilter	acceptNode()	Specifies whether a node is accepted or rejected. This interface is used by the NodeIterator and TreeWalker interfaces. This method returns one of the following values: <ul style="list-style-type: none"> • FILTER_ACCEPT - The current node is included. • FILTER_SKIP - The current node is not accepted, but the children of the current node are considered for acceptance. • FILTER_REJECT - The current node is not accepted. For TreeWalker methods, the children of the current node are not considered for inclusion.

Examples

The following `sampleFilter()` code example uses the `NodeFilter` interface. The implementation skips all `title` elements and rejects all `list` elements:

```
public class SampleFilter implements NodeFilter {
```

```

public short acceptNode (Node n) {
    if (n.getNodeType() == Node.ELEMENT_NODE) {
        Element elem = (Element) n;

        if ( elem.getNodeName().equals("title")) {
            return FILTER_SKIP;
        }

        if ( elem.getNodeName().equals("list")) {
            return FILTER_REJECT;
        }
    }

    return FILTER_ACCEPT;
}
}

```

Creating a `NodeIterator` object that uses the filter from the `sampleFilter()` example requires getting a handle to the `DocumentTraversal` implementation, as follows:

```
DocumentTraversal docTraversal = XhiveDriverFactory.getDriver().getDocumentTraversal();
```

The `createNodeIterator()` method is used to create a `NodeIterator` object and can use the following parameters:

- `root` - The node at which to start the traversal.
- `whatToShow` - The flag specifying which node types to include.
- `filter` - The filter to use. The value can be set to `null` if no filter is used.
- `entityReferenceExpansion` - A flag specifying whether to expand entity reference nodes.

The following example code traverses the first chapter of a document using a `NodeIterator` object and without a `NodeFilter` object.

```

System.out.println("\n#NodeIterator without a NodeFilter:");
NodeIterator iter = docTraversal.createNodeIterator(resultGetDocument,
    NodeFilter.SHOW_ALL,
    null,
    false);
Node node;

while ((node = iter.nextNode()) != null){
    System.out.println("    Node Name = " + node.getNodeName());
}

```

To restrict the traversal and not include `title` or `list` elements, change the second line of the previous example to:

```

NodeIterator iter = docTraversal.createNodeIterator(resultGetDocument,
    NodeFilter.SHOW_ALL,
    sampleFilter,
    false);

```

The following code instantiates the `sampleFilter` object:

```
NodeFilter sampleFilter = new SampleFilter();
```

Traversing a document using a `TreeWalker` object with a `sampleFilter` object instead of a `NodeIterator` skips the child nodes of the `list` element, as described in the following example code.

```
TreeWalker walker = docTraversal.createTreeWalker(resultGetDocument,
    NodeFilter.SHOW_ALL,
    sampleFilter,
    false);

while ((node = walker.nextNode()) != null){
    System.out.println("    Node Name = " + node.getNodeName());
}
```

Related topics

[Retrieving documents using DOM operations](#)

Traversing using function objects

Function objects are an elegant way to traverse documents because they enable separation and reusing traversal and function methods. A function object is a class that implements the `com.xhive.dom.interfaces.XhiveFunctionIf` interface. Define the following methods in a function object class:

- `isDone()`, which indicates whether the traversal can be terminated.
- `process()`, which contains the code for the actual processing of the node.
- `test()`, which indicates whether the node has to be processed with `process()`.

To create a function object that only processes `Element` nodes with an attribute `number`, you can declare the `test()` method as follows:

```
public boolean test (Node node){
    return node.getNodeType() == Node.ELEMENT_NODE && (
        (Element)node).hasAttribute("number");
}
```

The `test()` method is called for every node and determines whether to process the node.

The `process()` method of the function object class defines what has to happen with the nodes that pass the test as defined in `test()`:

```
public void process (Node node){
    String indentation = "";
    String elementName = ((Element)node).getTagName();
    if ( elementName.equals("article") ) {
        indentation = "  ";
    }

    System.out.println( indentation + elementName + " " + (
        (Element)node).getAttribute("number") );
}
```

The `isDone()` method, which the traversal method calls automatically, checks if the traversal has to continue or can terminate. In the example, all nodes have to be processed, so `isDone()` always returns `false`:

```
public boolean isDone (Node node){
```

```

        return false;
    }

```

You could, for example, use the `isDone()` method to limit the number of processed nodes to a specified number:

```

public boolean isDone (Node node){
    return nrResults == maxNrResult;

    // nrResults is incremented in process()
}

```

The `com.xhive.dom.interfaces.XhiveNodeIf` interface contains the traversal methods for function objects. The following traversal methods are available:

- `traverseAllNodesDocumentOrder()` traverses the current node and all descending nodes, including attributes in document order.
- `traverseAncestors()` traverses the ancestors of the current node.
- `traverseAttributesDocumentOrder()` traverses the attributes of the current node and its descending nodes in document order.
- `traverseBreadthFirst()` traverses the current node and all its descendants in breadth-first order.
- `traverseChildren()` traverses the children of the current node.
- `traverseDocumentOrder()` traverses the current node and all its descendants in document order.
- `traverseReverseDocumentOrder()` traverses the current node and all its descendants in reverse document order.

The following example uses the function `MyNumberFinder` to traverse all nodes within a library in document order and breadth-first:

```

MyNumberFinder numberFinder = new MyNumberFinder();

System.out.println("# traverse the charter library with traverseDocumentOrder:");
charterLib.traverseDocumentOrder(numberFinder);

System.out.println("# traverse the charter library with traverseBreadthFirst:");
charterLib.traverseBreadthFirst(numberFinder);

```

You could also use the same function object to traverse a single document in various ways:

```

Document chapter5Document = (Document)charterLib.get("UN Charter - Chapter 5");

System.out.println("# traverse \"UN Charter - Chapter 5\"
with traverseDocumentOrder:");
((XhiveDocumentIf)chapter5Document).traverseDocumentOrder(numberFinder);

System.out.println("# traverse \"UN Charter - Chapter 5\"
with traverseReverseDocumentOrder:");
((XhiveDocumentIf)chapter5Document).traverseReverseDocumentOrder(numberFinder);

```

Exporting XML documents

DOMs, such as a string or an output stream, can be serialized using the `toXml(...)` method in the `XhiveNodeIf` interface, or a `LSSerializer` DOM Load/ Save object, as described in the following example.


```
LSSerializer writer = charterLib.createLSSerializer();  
writer.getDomConfig().setParameter("format-pretty-print", Boolean.TRUE);  
String output = writer.writeToString(firstDocument);
```

Samples

[DOMLoadSave.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveNodeIf](#)

[org.w3c.dom.ls.LSSerializer](#)

Publishing XML documents

The `com.xhive.util.interfaces` package provides the following interfaces for publishing XML documents from xDB:

- **XhiveTransformerIf** for publishing using XSLT
- **XhiveFormatterIf** for publishing to PDF format

XSLT

xDB contains an XSL Transformation (XSLT) engine. XSLT can transform an XML source tree into any required result tree and publish XML content in (X)HTML, PDF and other formats. For more information about XSLT, see the [W3C website](#).

Publishing from xDB with XSLT requires an XML and XSL document within a Java application that uses the **transformToString()**, **transformToStream()**, or **transformToDocument()** method. Both the XML and XSL documents are stored in the database. The output can be another XML document, or a document of any other format.

The XSL document, which is actually an XML file, specifies the transformations that produce the desired output.

Note: When parsing XSL documents, the **XhiveLibraryIf.PARSER_NAMESPACES_ENABLED** option value must be `TRUE`. Otherwise an exception is thrown during transformation of the XML document.

PDF

XML documents can be published to PDF format with the **formatAsPDFToStream()** method in the **com.xhive.util.interfaces.XhiveFormatterIf** interface. This method can format either an XSL-FO

document or an XML document as a PDF string. Transforming an XML document also requires an XSL document.

Samples

[Publish2HTML.java](#)

[Publish2PDF.java](#)

API documentation

[com.xhive.util.interfaces.XhiveTransformerIf](#)

[com.xhive.util.interfaces.XhiveFormatterIf](#)

XLink interfaces

The XLink information is accessible through the DOM API because it is stored as attributes. The **com.xhive.dom.xlink.interfaces** package provides methods and interfaces that are more convenient and easier to use. A selection of these methods is listed in table [XLink methods, page 122](#). The **DomLinkBase.java** sample uses several of the XLink methods available in xDB.

Table 16 XLink methods

Methods	Description
getTitle(), getHref(), getRole()	Retrieve specific attributes.
getLinks(), getLinksByTitle(), getLinksByRole()	Retrieve all available links, by title, or role.
getNodesLinkingTo()	Retrieve all nodes linking to a specific resource.
getArcs(), getFrom(), getTo(), getStartingResources(), getEndingResources()	Retrieve all information from arcs.
getLocators(), getRole(), getLabel()	Retrieve all information from locators.
expandDocument(), getResources-LinkedBy()	Retrieve the content of the targeted resources.

Samples

[XLink.java](#)

[DomLinkBase.java](#)

API documentation

[com.xhive.dom.xlink.interfaces](#)

Using versioning

xDB offers versioning for documents and BLOBs. The **makeVersionable()** method of the **XhiveLibraryChildIf** interface controls versioning.

The example below uses the `makeVersionable()` method to enable versioning for the "briefing.xml" document.

```
XhiveLibraryChildIf doc = briefingLib.get("briefing.xml");
doc.makeVersionable();
```

The `getXhiveVersion()` method in the **XhiveLibraryChildIf** interface can be used to get the last version of a document:

```
// get the last version of doc
XhiveVersionIf lastVersion = doc.getXhiveVersion();
```

The **XhiveVersionIf** interface contains several methods for obtaining version information, including `getDate()`, `getLabel()` and `getCreator()`:

```
System.out.println("id: " + version.getId());
System.out.println("creation date: " + version.getDate().toString());
System.out.println("label: " + version.getLabel());
System.out.println("created by: " + version.getCreator().getName());
```

Samples

[Versioning.java](#)

[Branching.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveLibraryChildIf](#)

[com.xhive.versioning.interfaces.XhiveBranchIf](#)

[com.xhive.versioning.interfaces.XhiveCheckoutIf](#)

[com.xhive.versioning.interfaces.XhiveVersionIf](#)

[com.xhive.versioning.interfaces.XhiveVersionSpaceIf](#)

Working with versioned documents

A versioned document is a read-only document and cannot be modified until it has been checked out of the database using the `checkout()` method.

The following code example checks out a document.

```
// do a check out of the last version
Document lastVersionDoc = lastVersion.checkout();
```

If a user or an application attempts to modify a versioned document that has not been checked out, the system generates a `VERSION_ACCESS_DENIED` exception. A versioned document can only be checked out by one user at a time and the document is locked while it is checked out. If another user tries to check out the same document the system generates a `VERSION_CHECKED_OUT` exception.

Checked out documents can be checked in using the `checkIn()` method, as described in the following example.

```
// do some updates on "checkedOutDoc"
// ...
// check the document in
lastVersion.checkIn(lastVersionDoc);
```

The `checkIn()` method creates a version and releases the checkout lock. The checkout lock can also be released by using the `abort()` method. The `abort()` ignores all document changes and reverts back to the current version. It is not necessary to check in the specific library child that the checkout operation created. Any document or BLOB that is checked in creates another version.

Note: Versioned documents remain accessible for regular document retrieval. The last stored version of a versioned document is used for the retrieval, traversal, query, or index. Earlier versions are only searchable if they were created with a 'queryable' option.

Retrieving previous document versions

There are several methods to retrieve document versions, as described in the [Retrieval interfaces and methods, page 124](#) table.

Table 17 Retrieval interfaces and methods

Interface	Method	Description
XhiveLibraryChildIf	<code>getXhiveVersion()</code>	Returns the last version of a document.
XhiveVersionIf	<code>getPreviousVersion()</code>	Returns the previous version of a document.
XhiveVersionIf	<code>getNextVersions()</code>	Returns all subsequent versions of a document.
XhiveVersionIf	<code>getVersionSpace()</code>	Returns the version space of a document. Every versioned document has its own version space which is automatically created when versioning is enabled.

Example

The following example uses the `getVersionSpace()` method in **XhiveVersionIf** to access the version space of a document.

```
XhiveVersionSpaceIf versionSpace = doc.getXhiveVersion().getVersionSpace();
```

To retrieve a version by version space, use either the `getVersionById()` or the `getVersionByLabel()` method, as described in the following example.

```
// example of accessing a version via the getVersionById() method
XhiveVersionIf version1_1 = versionSpace.getVersionById("1.1");
Document doc1_1 = version1_1.getAsDocument();
```

Branching methods

xDB offers various methods for creating branches and for retrieving branch and version information, including:

Table 18 Branch and version methods

Interface	Method	Description
XhiveVersionIf	<code>createBranch()</code>	Creates a new branch.

Interface	Method	Description
XhiveVersionSpaceIf	getBranches()	Returns the branches of a document.
XhiveBranchIf	getVersions(), getVersionByDate(), getHeadLibraryChild()	Returns the version and documents of a branch.

Node-level versioning

Instead of checking out the entire document, users can check out individual document nodes and all their descendants.

The following example code checks out a document node.

```
XhiveDocumentIf doc = ...; // Some versioned document
Node introChapter = doc.executeXQuery("/root/chapter[@id='intro']").next().asNode();
XhiveVersionIf version = doc.getXhiveVersion();
XhiveBranchIf branch = version.getBranch();
// Create an owner document for the copy of the chapter to be edited
XhiveDocumentIf temporaryDoc = session.createTemporaryDocument();
XhiveCheckoutIf checkout = branch.checkoutNode(introChapter, temporaryDoc);
Node chapterCopy = checkout.getNodeCopy();
// Edit the copy of the chapter contained in chapterCopy
// ...
Map<Node, Node> nodes = Collections.singletonMap(introChapter, chapterCopy);
branch.checkInNodesAndMetadata(nodes, null, 0);
```

The following example code checks out metadata fields. Checking out a particular key name allows checking in a value for that key.

```
XhiveLibraryChildIf lc = ...; // Some versioned document or blob
XhiveVersionIf version = lc.getXhiveVersion();
XhiveBranchIf branch = version.getBranch();
branch.checkOutMetadataField("key");
String oldValue = lc.getMetadata().get("key"); // If required
Map<String, String> metadata = Collections.singletonMap("key", "newValue");
branch.checkInNodesAndMetadata(null, metadata, 0);
```

Any number of nodes and metadata fields can be checked in at once to create a single new version of the document in that branch. Because nodes are checked out from a branch, the checkin creates another head version of the branch, regardless of the latest version.

Note: The `XhiveVersionIf.createBranch()` method creates another branch.

Using searchable versions

To enable historical searching of a versioned library child, you must mark it as queryable when you create it.

The example below uses the `makeVersionable(boolean queryable)` method to create the "briefing.xml" document with version searching enabled.

```
XhiveLibraryChildIf doc = briefingLib.get("briefing.xml");
doc.makeVersionable(true);
```

To determine whether an existing version is searchable, you can use the `isQueryable()` method of the `XhiveVersionIf` interface. This is accessible through `XhiveLibraryChildIf.getXhiveVersion()`.

Querying

For queryable documents, xDB provides XQuery `xhive:collection-*-date` extension functions. For more information, refer to the section about [XQuery extension functions](#).

Indexing

Default indexes only store info about the current version of the document. On documents that were created with the 'queryable' option set to true, you can use `XhiveIndexIf.VERSION_INFO` to enable indexing of old versions. On multipath indexes, which don't use the `XhiveIndexIf` options, you can use `XhiveExternalIndexConfigurationIf.setStoreVersionInfo(true)`.

API documentation

com.xhive.dom.interfaces.XhiveLibraryChildIf

Using metadata on library children

Library (libraries, documents and blobs) can have metadata, consisting of key/value pairs. Such metadata can be used for various purposes, for example for storing information about a document if that data does not fit the DTD of the document. Metadata can be indexed and queried.

The metadata is a `java.util.Map` that can only contain strings. Example:

```
XhiveDocumentIf doc = ...;
doc.getMetadata().put("author", "Jane Doe");
```

When a new version of a versioned document or blob is checked in, the metadata of the new version is also checked in, overriding the previous version of the metadata.

Using abstract schemas

Abstract schemas contain interfaces for handling schema information, such as the structure of element declarations, and interfaces for applying schema information to DOMs validation. xDB provides full abstract schema support for DTDs and a more limited support for XML schema, with some product-specific modifications. For a detailed description of the interfaces, see the API documentation. **Note:** The W3C has canceled the DOM L3 abstract schema specifications. xDB continues to support the AS specification for model manipulation.

xDB stores the abstract schema of a document in a catalog during validated parsing. For more information, see the chapter on [catalogs and validation, page 221](#).

The following example shows how to use the `setActiveASModel()` method with an abstract schema that has already been stored in the catalog.

```
Document doc = charterLib.createDocument(null, "chapter", null);
DocumentAS document = (DocumentAS) doc;
document.setActiveASModel(model);
```

The `ASModel`, `ASElementDecl`, `ASAttributeDecl`, `ASNotationDecl`, and `ASEntityDecl` interfaces get information about the declarations in a DTD. These interfaces do not work with XML Schemas. The XML Schema API must be used to obtain information from XML schemas.

Examples

The following code example gets all required attributes of an element.

```
ASModel model = ((DocumentAS) document).getActiveASModel();
ASElementDecl eltDeclaration = model.getElementDecl(element.getNamespaceURI(),
    element.getTagName());
ASNamedObjectMap attributeDeclarations = eltDeclaration.getASAttributeDecls();
System.out.println("The following attributes are all required:");
for (int i = 0; i < attributeDeclarations.getLength(); i++) {
    ASAttributeDecl attDecl = (ASAttributeDecl) attributeDeclarations.item(i);
    // Check whether this attribute is required
    if (attDecl.getDefaultType() == ASAttributeDecl.REQUIRED) {
        String attName = attDecl.getObjectName();
        System.out.println(attName);
    }
}
```

The following example code shows how to parse a DTD or XML Schema directly into the catalog of a library.

```
ASDOMBuilder builder = (ASDOMBuilder) charterLib.createLSParser();
model = builder.parseASURI(url, ASDOMBuilder.DTD_SCHEMA_TYPE);
unCharterCatalog.setPublicId(publicId, model);
```

The following example code serializes a schema model in the catalog.

```
DOMASWriter writer = (DOMASWriter) charterLib.createLSSerializer();
ByteArrayOutputStream output = new ByteArrayOutputStream();
writer.writeASModel(output, model);
String schemaString = output.toString();
```

API documentation

org.w3c.dom.as

com.xhive.dom.interfaces.XhiveCatalogIf

Using the APIs for XSL transformations

The `XhiveTransformerIf` and `XhiveFormatterIf` interfaces in `com.xhive.util.interfaces` are convenient access methods to Xalan and FOP of Apache project respectively. In some cases, use the more advanced options of these packages. The examples below provide a starting point for using these interfaces.

XhiveFormatterIf.formatAsPDF

```
import org.apache.avalon.framework.logger.ConsoleLogger;
```

```

import org.apache.avalon.framework.logger.Logger;
import org.apache.fop.apps.Driver;
import org.apache.fop.messaging.MessageHandler;

public static void formatAsPDFToStream(DOMImplementation docCreator,
Document xmlSource, Document xslSource, OutputStream os)
    throws XhiveException {
    try {
        // XSLT part
        XhiveTransformerIf transformer = XhiveDriverFactory.getDriver().
            getTransformer();
        Document foDocument = transformer.transformToDocument(docCreator,
            xmlSource, xslSource);

        // FOP part
        Driver driver = new Driver();
        Logger logger = new ConsoleLogger(ConsoleLogger.LEVEL_ERROR);
        MessageHandler.setScreenLogger(logger);
        driver.setLogger(logger);
        driver.setRenderer(Driver.RENDER_PDF);
        driver.setOutputStream(os);
        driver.render(foDocument);
    } catch (Exception e) {
        //e.printStackTrace();
        throw new XhiveException(XhiveException.FORMAT_EXCEPTION, e);
    }
}

```

XhiveTransformerIf.transform

```

import com.xhive.core.interfaces.XhiveSessionIf;
import com.xhive.error.XhiveException;
import com.xhive.dom.*;
import org.w3c.dom.*;
import javax.xml.transform.*;
import java.io.*;
import java.util.Iterator;

public Document transformToDocument(DOMImplementation docCreator,
Node xmlSource, Document xslSource)
    throws XhiveException {

    Document result = docCreator.createDocument("", "Result", null);
    Node rootElem = result.getDocumentElement();
    result.removeChild(rootElem);

    transform(xmlSource, xslSource, new DOMResult(result));
    return result;
}

public void transformToStream(Node xmlSource, Document xslSource,
    Writer writer)
    throws XhiveException {
    transform(xmlSource, xslSource, new StreamResult(writer));
}

public String transformToString(Node xmlSource, Document xslSource)
    throws XhiveException {

```



```

        StringWriter result = new StringWriter();
        transformToStream(xmlSource, xslSource, result);
        return result.toString();
    }

private void transform(Node xmlSource, Document xslSource, Result result) {
    try {
        // Try to initialize URI resolver
        URIResolver myResolver;
        try {
            XhiveSessionIf session = ...; // Left as an exercise to the reader
            if (session != null) {
                myResolver = new XhiveURIResolver(session);
            } else {
                myResolver = null;
            }
        } catch (Exception e) {
            // (No session?) Fine, then we don't use a URIResolver
            myResolver = null;
        }

        TransformerFactory tFactory = TransformerFactory.newInstance();
        if (myResolver != null) {
            // Factory resolver must be set before transformer is created
            tFactory.setURIResolver(myResolver);
        }
        Transformer transformer = tFactory.newTransformer(new DOMSource(
            xslSource));
        if (myResolver != null) {
            transformer.setURIResolver(myResolver);
        }

        transformer.transform(new DOMSource(xmlSource), result);
    } catch (Exception e) {
        throw new XhiveException(XhiveException.TRANSFORM_EXCEPTION, e);
    }
}

/**
 * URI resolver that translates
 * xhive:path#query
 * into an xquery run on path, e.g.
 * xhive:/plays#//TITLE
 */
private class XhiveURIResolver implements URIResolver {
    private static final String XHIVE_PREFIX = "xhive:";
    private static final String SEPARATOR = "#";

    private XhiveSessionIf session;

    public XhiveURIResolver(XhiveSessionIf session) {
        this.session = session;
    }

    public Source resolve(String href, String base) throws TransformerException {
        // Do we need to do anything?

```

```

    if (((base == null) || (!base.startsWith(XHIVE_PREFIX))) &&
        (!href.startsWith(XHIVE_PREFIX))) {
        return null;
    } else {
        // Process href
        // Up to us to come up with a result
        if (!href.startsWith(XHIVE_PREFIX)) {
            // Create href with base
            href = base + href;
        }
        // Strip xhive: from href
        href = href.substring(XHIVE_PREFIX.length());
        if (!href.startsWith("/")) {
            href = "/" + href;
        }

        // Separate in path and query
        String path = null;
        String query = null;
        if (href.indexOf(SEPARATOR) != -1) {
            path = href.substring(0, href.indexOf(SEPARATOR));
            query = href.substring(href.indexOf(SEPARATOR) + 1);
        } else {
            path = href;
        }
        // Get query context
        XhiveLibraryChildIf contextNode = session.getDatabase().
            getRoot().getByPath(path);
        if (contextNode == null) {
            // Nothing found, error or null?
            throw new TransformerException("XhiveXalanTransformer:
                Could not resolve " + href);
            //return null;
        } else {
            if (query == null) {
                if (contextNode instanceof XhiveDocumentIf) {
                    return new DOMSource(contextNode);
                } else if (contextNode instanceof XhiveBlobNodeIf) {
                    return new StreamSource(((XhiveBlobNodeIf) contextNode).
                        getContents());
                } else {
                    throw new TransformerException("XhiveXalanTransformer: "
                        + href + " is not a document");
                }
            } else {
                return getExecuteQuerySource(contextNode, query, href);
            }
        }
    }
}

private Source getExecuteQuerySource(XhiveLibraryChildIf contextNode,
String query, String href) throws TransformerException {
    if (query.startsWith("xpointer(") && query.endsWith(")")) {
        query = query.substring("xpointer(".length(), query.length() - 1);
    }
    Iterator queryResult = null;
    try {
        queryResult = contextNode.executeXQuery(query);
    }
}

```

```

    } catch (XhiveException e) {
        throw new TransformerException("XhiveXalanTransformer:
            Problem with query " + href + ": " + e.getMessage(), e);
    }
    // We will only use the first result here (otherwise we would
    // have to include a new top-element)
    if (queryResult.hasNext()) {
        XhiveXQueryValueIf queryValue = (XhiveXQueryValueIf)
            queryResult.next();
        // Is it a node?
        try {
            return new DOMSource(queryValue.asNode());
        } catch (XhiveException e) {
            // must be XQUERY_ERROR_VALUE, so interpret it as a string
            return new StreamSource(new StringReader(queryValue.asString()));
        }
    } else {
        throw new TransformerException("XhiveXalanTransformer:
            Query " + href + ": " + " has no results");
    }
}
}
}

```

For improved Xalan XSLT performance, it is best to create `Templates` objects using the `newTemplates` call on `TransformerFactory`. The example below shows how to use the interface. For more information, see the Xalan documentation.

```

TransformerFactory tFactory = TransformerFactory.newInstance();
tFactory.newTemplates(new DOMSource(xslSource));
// Now keep this translet cached somewhere, and for transforming do:
Transformer transformer = translet.newTransformer();
transformer.transform( new DOMSource(xmlSource), new StreamResult(writer) );

```

One advantage of a compiled stylesheet is that it no longer has references to any xDB persistent data, so you can use the compiled version with any session.

Session and Transaction Management

This chapter contains the following topics:

- [Sessions, transactions and locking](#)
- [Namebase and locking](#)
- [Session lifecycle](#)
- [Referencing database objects in sessions](#)
- [Multithreaded session handling](#)
- [Transaction isolation in sessions](#)
- [Managing locking conflicts](#)
- [Read-only transactions](#)
- [Getting info on sessions and locks](#)

Sessions, transactions and locking

xDB accesses the database using transactions within *sessions*, which are connected to the database. A session can have multiple transactions that can perform data retrievals, changes or rollbacks on the database.

Whenever data is modified during a transaction, xDB locks that data to ensure that it cannot be modified by other transactions. The locks are released when the locking transaction is committed or rolled back.

These locks are placed on the related objects as soon as they are modified, and they are released after a commit or rollback call. As long as an object remains locked, other transactions cannot change it. By default, a transaction that tries to use a locked object is blocked until the lock is released.

In addition to such implicit data locking during modification, developers can lock libraries explicitly. To obtain debug information about open transactions and their associated locks from the command line, you can use the [`xdb info` command](#), [page 251](#).

An xDB database is divided into *locking contexts*. A locking context represents a group of objects that are locked together, which means that locking one of the objects in the group locks all the other objects in the group at the same time. As soon as anything within a locking context is changed, the entire locking context is locked. For example, a library is a locking context. When a user adds a document to a library, other concurrent sessions cannot add any other documents to that library until the commit is performed.

The [Locking behavior](#), [page 134](#) table describes several locking scenarios.

Table 19 Locking behavior

Action	Locked
Add/remove a document (or library)	The library to which the document is added.
Modify a document	The document.
Add/remove an index to/from a library	The library to which the index is added.
Add/remove a user/group	The database object (which means that one concurrent thread can make these changes).
Update a user/ group	The database object.
Update a context conditioned index	The context conditioned index.

Namebase and locking

Documents stored in xDB use an internal data structure called *namebase*. This structure is relevant to the locking behavior but cannot be accessed directly using the API. The namebase maps element and attribute names to small integers which are processed faster. The namebase is locked when it is modified.

When a library is created, the following option influences the namebase locking behavior:

- **XhiveLibraryIf.LOCK_WITH_PARENT**

By default, each library is created with its own namebase. Using more namebases improves concurrency, but adds some space and processing overhead. In some cases, for example if there are many libraries with little content, it may be better for a new library to share the namebase of the parent library.

API documentation

com.xhive.dom.interfaces.XhiveLibraryIf

Session lifecycle

In xDB, the data is accessed using transactions within sessions. A transaction starts with a call to `session.begin()` and ends with a call to `session.commit()` or `session.rollback()`. When a session is in a transaction the data in the database can be viewed or changed.

You can use a call to `session.isOpen()` to check if a session is in a transaction (an open or active session is a session in a transaction).

At a minimum, a full session lifecycle consists of the following operations:

- `createSession()`, page 141

- [connect\(\)](#), page 142
- [begin\(\)](#), page 142
- perform database actions, optionally using [checkpoint\(\)](#), page 142
- [commit\(\)](#), page 142 or [rollback\(\)](#), page 142
- [disconnect\(\)](#), page 143
- [terminate\(\)](#), page 143

Follow this model closely when creating xDB applications.

When a transaction does not need to make changes to the data then make the session read-only. Do this with `session.setReadOnlyMode()`. Read-only transactions do not need locks. A session can enable or disable the read-only mode when not in a transaction.

After ending a transaction you can start a new transaction. You end a transaction with `session.commit()` or `session.rollback()`.

After a call to `session.disconnect()` you can do a new call to `session.connect()` and then start a new transaction.

After a call to `session.terminate()` you can not use the session anymore.

The code below is a complete example how to use sessions following this model:

```
package com.emc.xhivesupport.demo;

import com.xhive.XhiveDriverFactory;
import com.xhive.core.interfaces.XhiveDriverIf;
import com.xhive.core.interfaces.XhiveSessionIf;
import com.xhive.dom.interfaces.XhiveLibraryIf;

public class SimpleMain {
    public static void main(String[] dummy) {
        XhiveDriverIf driver = XhiveDriverFactory.getDriver();
        driver.init(1024);
        XhiveSessionIf session = driver.createSession();
        session.setReadOnlyMode(true); // if no write access needed
        try {
            session.connect("Administrator", "northsea", "united_nations");
            session.begin();
            // your code goes here
            session.commit();
        } finally {
            session.rollback();
            session.terminate();
        }
        driver.close();
    }
}
```

Whenever exceptions occur in a transaction, the session must always rollback, whereas whenever a transaction succeeds, the session must always commit. This is also true when the session is in

read-only mode. Even though there are no changes, and no locks to release, a read-only transaction still does eat resources. A commit or rollback is needed to free these resources.

The `finally` block is always executed. If there are exceptions in the `try` block, the `session.commit()` in the `try` is not executed and the `session.rollback()` in the `finally` block will rollback the session. If there are no exceptions, the execution of the `session.rollback()` immediately follows the `session.commit()`. In this case the rollback simply does nothing.

A `session.terminate()` is allowed when a session is not in a transaction. No need to replace the above `finally` block by this:

```
    } finally {
    if (session.isOpen()) {
        session.rollback();
    }
    if (session.isConnected()) {
        session.disconnect();
    }
    session.terminate();
}
```

Following this model, you can always create your sessions in method scope, and usually you can do all the session management in the same method. It is not recommended to 'remember' sessions as instance members or class members. It will almost certainly lead to errors related to not respecting the sessions lifecycle.

Samples

[UseSessions.java](#)

API documentation

[com.xhive.core.interfaces.XhiveSessionIf](#)

Joined sessions and session pools

Joined sessions

You can have concurrent transactions: different threads can each create their own session and execute transactions concurrently. Here, locks make sure data cannot be changed in conflicting ways. But to make sure that data cannot be changed in conflicting ways, you can not do different things concurrently in one transaction.

The requirement that a session is used by one thread at a time is ensured by the xDB session API as follows:

- With `session.join()` you can register the current thread to the session (the session joins the current thread).
- With `session.leave()` you can unregister the current thread from the session (the session leaves the current thread).
- With `session.isJoined()` you can check whether the current thread is registered to the session (the session is joined to the current thread).

Almost all operations on the session require `session.isJoined() == true`. In particular `session.leave()` requires `session.isJoined() == true`. To make sure that one thread can not hijack a session that is joined to some other thread `session.join()` requires the session is not joined to any thread. As a consequence, if a session is joined to some other thread, and that other thread is somehow gone, there is nothing you can do about it.

Notice, `driver.createSession()` implicitly joins the returned session to the current thread. That explains why you usually do not need to bother about this.

Working with a Sessionpool

In early versions of xDB the create and terminate of a (remote) session was expensive, because of the create and close of connections under the hood. It was recommended to avoid these operations as much as possible by using a session pool. Starting with xDB 8, xDB has a connection manager, keeping its own pool of connections. Unless performance tests or special conditions suggest otherwise, in production code, session pools usually are not needed.

If you still need a session pool, a session returned to the pool by one thread will in general be reused by some other thread. The session should leave the current thread when it is returned to the pool. The leave and join of sessions is hard enough to make sure it is coded only once, as part of the implementation of the pool and not on each use in the code using the pool.

The example below shows how you could implement a session pool:

```
package com.emc.xhivesupport.demo;

import com.xhive.core.interfaces.XhiveDriverIf;
import com.xhive.core.interfaces.XhiveSessionIf;

import java.util.concurrent.ConcurrentLinkedQueue;

public class SessionPool {
    private XhiveDriverIf driver;
    private ConcurrentLinkedQueue<XhiveSessionIf> freeSessions =
        new ConcurrentLinkedQueue<XhiveSessionIf>();    // pool implementation

    // singleton
    private static SessionPool instance = new SessionPool();
    private SessionPool() {
    }
}
```

```
public static SessionPool getInstance() {
    return instance;
}

// core functionality
public synchronized void init(XhiveDriverIf driver) { // thread safe
    this.driver = driver;
}
public XhiveSessionIf getSession() {
    XhiveSessionIf session = freeSessions.poll(); // thread safe
    if (session == null) {
        session = driver.createSession();
    } else {
        session.join();
    }
    return session;
}
public void returnSession(XhiveSessionIf session) {
    session.rollback();
    session.disconnect();
    session.leave();
    freeSessions.add(session); // thread safe
}
public synchronized void close() { // thread safe
    XhiveSessionIf session = freeSessions.poll();
    while (session != null) {
        session.terminate();
        session = freeSessions.poll();
    }
    instance = null; // encourage garbage collection
}
}
```

The example below shows how to use the session pool:

```
package com.emc.xhivesupport.demo;

import com.xhive.XhiveDriverFactory;
import com.xhive.core.interfaces.XhiveDriverIf;
import com.xhive.core.interfaces.XhiveSessionIf;

public class TestSessionPool implements Runnable {
    public static void main(String[] dummy) throws InterruptedException {
        XhiveDriverIf driver = XhiveDriverFactory.getDriver();
        driver.init(1024);
        SessionPool pool = SessionPool.getInstance();
        pool.init(driver);
        Thread[] threads = new Thread[100];
        for (int i = 0; i < threads.length; i++){
            threads[i] = new Thread(new TestSessionPool());
            threads[i].start();
        }
    }
}
```

```

    for (int i = 0; i < threads.length; i++) {
        myJoin(threads[i]);
    }
    pool.close();
    driver.close();
}
private static void myJoin(Thread t) {
    try {
        t.join();
    } catch (InterruptedException x) {
        Thread.currentThread().interrupt();
    }
}
public void run() {
    SessionPool pool = SessionPool.getInstance();
    for (int i = 0; i < 20; i++) {
        XhiveSessionIf session = pool.getSession();
        session.setReadOnlyMode(true); // if no write access needed
        try {
            session.connect("Administrator", "northsea", "united_nations");
            session.begin();
            // your code goes here
            session.commit();
        } finally {
            pool.returnSession(session);
        }
    }
}
}
}

```

Notice, the use and implementation of this session pool is as follows:

- After a driver is initialised, it is passed to the pool.
- Whenever a session is needed, the user must get one from the pool.
- When you get a session from the pool, in `SessionPool.getSession()`, the pool must do the `session.join()`.
- The user must do the `session.connect()`, the `session.begin()` and, if a transaction was succesfull, the `session.commit()`.
- When you return a session to the pool, in `SessionPool.returnSession()`, the pool must do the `session.rollback()` if the transaction failed. Also in this method, the pool must do the `session.disconnect()` and the `session.leave()`. Notice, from the viewpoint of the pool, after return from this method the current thread is gone.
- All sessions must eventually be returned to the pool.
- The administration of free sessions must be thread-safe.
- After all sesssions are returned to the pool, the user must close the pool, and the pool must terminate all sessions.

Other requirements for a pool are conceivable. For instance, pools that keep connected sessions, or keep sessions that come from different drivers.

API documentation

com.xhive.core.interfaces.XhiveSessionIf

Sessions and references to database objects

Database object references

Objects in the database can only be accessed in an open transaction. Furthermore, objects retrieved from a database in one transaction cannot be used in a subsequent transaction. For example, the following code snippet will throw an `XhiveException` with errorcode `OBJECT_CLOSED`:

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
System.out.println(library.getNumChildren());
session.commit();

session.begin();
System.out.println(library.getNumChildren()); // ERROR: OBJECT_CLOSED
session.commit();
```

The reference `library` points to a database object. The line:

```
System.out.println(library.getNumChildren());
```

appears two times in the code. The first occurrence is in the same transaction where we get the `library` reference. The second occurrence is in another transaction.

Solution: just get the database object again:

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
System.out.println(library.getNumChildren());
session.commit();

session.begin();
library = session.getDatabase().getRoot(); // get again
System.out.println(library.getNumChildren()); // OK
session.commit();
```

Use of checkpoint

The `session.checkpoint()` method allows you to write code like this:

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
System.out.println(library.getNumChildren());
session.checkpoint(false); // or: session.checkpoint(true)
System.out.println(library.getNumChildren()); // ref library still valid
session.commit();
```

The effects of `void XhiveSessionIf.checkpoint(boolean downgradeLocks)` are:

- Commit all persistent operations executed since the last `begin()` or `checkpoint()`, and keep all locks acquired during the current transaction. The current transaction remains active.
- If `downgradeLocks == true`, downgrade write locks to read locks.

There may be use cases where you need checkpoints. For instance, you want to commit expensive operations one by one, but not lose all the work if an exception occurs, while subsequent operations on the other hand need a consistent view of all the committed work in progress.

The downside of using checkpoints is that locks are not released. Using `checkpoint()` makes your transactions longer and may hinder concurrency, and, as a consequence, can decrease throughput and overall performance. Usually it is better to have shorter transactions that release all their locks, using `commit()` or `rollback()`.

Hence, replacement of:

```
session.commit();
session.begin();
```

by:

```
session.checkpoint();
```

should only be done to support a specific use case.

API documentation

com.xhive.core.interfaces.XhiveSessionIf

XhiveDriverIf.createSession()

The `createSession()` method creates a new database session. The session is implicitly joined to the current thread.

Example

The following example creates a session.

```
XhiveDriverIf driver = ...; // e.g. from XhiveDriverFactory.getDriver()
```

```
XhiveSessionIf session = driver.createSession();
```

API documentation

com.xhive.core.interfaces.XhiveDriverIf

connect()

The connect() method connects a session to a database. The connect method has a relatively small overhead. This call's overhead is relatively small, so in a multi-user setting you may choose to connect every time you start using a session for an individual users.

begin()

The begin() method starts a transaction. All database changes are part of the transaction and only become visible in the database after a commit() call or a checkpoint() call. All data read from the database is in the same state as at the time of the begin() call.

checkpoint()

The checkpoint() method commits database changes within a transaction to the database, including all changes that were made since the begin() call or the last checkpoint() call. The changes are visible to other sessions. The transaction remains open after a checkpoint() call.

A checkpoint() call keeps the locks on the database, unless the **true** option is passed to downgrade the locks. A checkpoint() call is faster than a commit() call followed by a begin() call.

Like a commit() call, a checkpoint() call deletes all temporary objects, such as XQuery constructed result nodes or elements created but not appended to their document. Therefore, even though references to existing database objects remain valid, these temporary objects can no longer be used after a checkpoint() call.

For optimal concurrency, it best to keep the number of readlocks as low as possible. For this reason, it may be better to use a commit() call followed by a begin() call instead of a checkpoint() call.

commit()

The commit() method commits all data changes to the database, cleans up data temporarily stored in the database, and releases all locks.

The time to process a commit() method depends on the changes that were made in the transaction.

rollback()

The rollback() method revokes all changes in the database that were made since calling the begin() method or the checkpoint() method. If changes were already written to the disk, the rollback() method can have considerable overhead.

disconnect()

The `disconnect()` method returns a session back to the initial state it was in when the session was created. The `disconnect()` method has no overhead.

Note: Although a **`disconnect()`** call marks the end of a session scope, it does not free all the resources allocated by the session. To improve performance, use the **`terminate()`** method to terminate a session when it is no longer needed. If the session is a remote session, the TCP connection to the page server is returned to the connection manager. If the **`terminate()`** method is not called, the connection is closed when the session is finalized, after it has been garbage collected.

terminate()

The `terminate()` method closes the connection to the page server. Terminating a local session has no effect.

After calling the `terminate()` method, the session object can no longer be used.

If a session is not terminated, it continues to use resources until it is garbage collected. Open sessions in transaction can have internal references pointing to them that are otherwise never garbage collected. Sessions that are not in transaction are garbage collected when all references are released.

join()

The `join()` method joins a session to the current thread. Only the current thread can use database objects, for example documents that belong to this session.

When a session is created, it is automatically joined to the thread that creates it.

You should always call the `join()` method before you start using a session in a given thread. When working with servlets or EJBs, each request is executed in an 'unknown' thread, but during execution of the request the thread does not change, so `join()` is only called once. The time it takes to execute the `join()` method is almost negligible. Still, you should only call it as needed, as when it can help you to detect unexpected or unwanted thread changes.

Note: It is not possible to use a single session concurrently in multiple threads. Do not try to serialize the use of a session in multiple threads by synchronizing on the session object. This conflicts with internal use of the session synchronization and may lead to deadlocks. If it is necessary to serialize the use of a session, the synchronization should be done on an application object.

leave()

The `leave()` method unbinds a session from a thread.

It is important to call the `leave()` method on sessions that are no longer used in threads. Otherwise, it may become possible to terminate the session after the thread is exited and the `terminate()` method is called from another thread.

Referencing database objects in sessions

Objects in the database can only be accessed in an open transaction. Furthermore, objects retrieved from a database in one transaction cannot be used in the following transactions after a `commit()` call. For example, the following call is not allowed:

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
session.commit();
session.begin();
System.out.println(library.getName());
session.commit();
```

Instead, use the following call:

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
session.commit();
session.begin();
library = session.getDatabase().getRoot();
System.out.println(library.getName());
session.commit();
```

After calling the `commit()` and `begin()` methods, the library could have been removed in another session. An attempt to use an object in a different transaction usually generates an `XhiveException.OBJECT_DEAD` error. The error indicates that the object can no longer be used in the Java application.

Instead of calling the `commit()` and `begin()` method, call the `checkpoint()` method. The `checkpoint()` method applies permanent changes, does not refresh the database view, and keeps all locks. All references to database objects that were retrieved can still be used.

Multithreaded session handling

xDB accesses the database using transactions within sessions. Each session is created using `XhiveDriverIf.createSession()`.

Sessions can perform a variety of operations, such as:

- Read-only XQueries
- Read-Write XQueries
- Reading libraries using DOM API
- Storing documents using DOM API
- Alter existing documents

A session can be used by multiple threads, but by only one thread at a time. To free a session for use by the next thread, the current thread must call `leave()`, [page 143](#). To use the session, the next thread must then call `join()`, [page 143](#).

Operation execution

A good way to perform different operations consistently over multiple threads is to first create a base class for performing an operation, which would include the session management code, the exception handling code and so forth. Then create various derived classes whose only purpose is to run the desired operation.

In general, if you know in advance that the operation is read-only, it is best to set the session state to read-only. This may improve concurrency, because no locks are taken by [read-only transactions](#), [page 148](#).

Exception handling

Where operations happen concurrently, proper session exception handling is of the utmost importance, to help avoid data corruption, deadlocks and unexpected behaviour.

First and foremost: if an exception occurs, make sure you roll back the transaction:

```
try {
    session.begin()
    //Some sort of operation
    session.commit()
}

catch (Exception e) {
    e.printStackTrace();
    session.rollback();
}
```

When operations are being performed concurrently, locking issues may arise (for example: transaction A has a read lock on the database, while transaction B tries to retrieve a write lock on the database). In such cases a **XhiveLockNotGrantedException** will be raised. Since the issue can be temporary (for example, if you retry the offending operation later, the lock may have been released), a good way to handle such an exception may be to rollback the session and try again.

```
boolean completed = false;
int unsuccessfullAttempts = 0;

while (!completed && unsuccessfullAttempts < MAX_ATTEMPTS) {
    try {
        session.begin();
        executeOperation(session);
        session.commit();
        completed = true;
    } catch (XhiveLockNotGrantedException e) {
        unsuccessfullAttempts++;
        session.rollback();
    }
}
```

Samples

[MultithreadedOperations.java](#)

Transaction isolation in sessions

xDB supports transaction isolation and atomicity. When the **begin()** method is called, the database view for that session is updated so that changes made in other transactions within other sessions are visible.

The example below shows three parallel sessions: read-write session A and two read-only sessions B and C. Each column lists the actions in a session in chronological order. The table shows at what point a change made in read-write session A becomes visible to sessions B and C.

Table 20 Session calling **begin()** updates its database view

Session A (read-write)	Session B (read-only)	Session C (read-only)
begin()		
	begin()	
addDocument 'doc'		
	// 'doc' not visible	begin() // 'doc' not visible commit()
commit()	// 'doc' not visible commit()	begin() // ' doc ' is visible commit()
	begin() // ' doc ' is visible commit()	

The document 'doc' added in the transaction within session A remains invisible in any other transaction until the transaction within session A is committed. Even then, the open transactions of sessions B and C do not see the added document until their next **begin()**.

Managing locking conflicts

A transaction cannot continue if it tries to get a write lock on a database object while another transaction has a read lock (or a write lock) on the same object. A similar conflict occurs when one transaction tries to get a read lock on a database object while another transaction has a write lock. What happens depends on the **XhiveSessionIf.setWaitOption()** wait option setting and the status of the collection of all current locks. The default setting is **WAIT**.

By default, the transaction attempting to modify the object is blocked until the other transaction releases the lock. If the wait option is set to **NO_WAIT**, an **XhiveLockNotGrantedException** error is thrown as soon as a transaction encounters a locked object. It is also possible to specify a time interval

in milliseconds for a transaction to wait for a lock grant. If the wait time has passed and the other transaction still locks the database object, an `XhiveLockNotGrantedException` error is thrown.

Lock exceptions can occur regardless of the wait option setting. For example:

```
Transaction A reads document X
Transaction A writes document X
Transaction B reads document Y
Transaction A wants to write document Y
    -> blocks because transaction B already has a readlock
Transaction B wants to read X
    -> blocks because transaction A already has a writelock
```

At this point, both transactions are unable to continue, because each one is waiting for the other to finish. In this case, xDB picks one transaction and throws an `XhiveDeadlockException` error, which is a subclass of the `XhiveLockNotGrantedException` error. If a rollback is performed on that transaction, the locks are released so that the other transaction can continue.

It is possible to get a deadlock even when only one database object is involved. For example:

```
Transaction A reads document X
Transaction B reads document X
Transaction A wants to write document X
    -> blocks because transaction B already has a readlock
Transaction B wants to write document X
    -> blocks because transaction A already has a readlock
```

Any application code should always take into account that locking exceptions can occur. Usually the course of action is to rollback the transaction and retry the same transaction a number of times:

```
public static void main(String[] dummy) {
    XhiveDriverIf driver = XhiveDriverFactory.getDriver();
    driver.init(1024);
    XhiveSessionIf session = driver.createSession();
    session.setReadOnlyMode(false); // read-only transactions need no locks
    try {
        session.connect("Administrator", "northsea", "united_nations");
        session.begin();
        int numAttempts = 0;
        boolean committed = false;
        while (numAttempts < 5 && !committed) {
            try {
                // your transaction code goes here
                session.commit();
                // not again after commit
                committed = true;
            } catch (XhiveLockNotGrantedException x) {
                // try again
                session.rollback();
                session.begin();
                numAttempts++;
            }
        }
    } finally {
        session.rollback();
        session.terminate();
    }
    driver.close();
}
```

```
}
```

Note: Read-only transactions do not need locks, so they do not need such retry logic. Use read-only transactions whenever you can.

Read-only transactions

By default, transactions can modify database objects. Transactions are made read-only by calling the `setReadOnlyMode()` method. Read-only transactions cannot modify database objects. The advantage of read-only transactions is that they do not take any locks. This method improves concurrency with transactions that modify data.

To get a consistent view of the database without using locks, read-only transactions view a logical snapshot of the data at the time the transaction begins. Read-only transactions do not recognize any modifications to the data.

Data pages that have been deleted are not reallocated to new documents as long as there are still open transactions that could use the old data. Therefore, transactions should not be kept open indefinitely.

The `checkpoint()` method does not affect read-only transactions.

Getting info on sessions and locks

You can use the `XhiveDriverIf.printSessionInformation()` API to obtain debug information about open transactions and their associated locks. From the command line, you can use the **xdb info command**, [page 251](#). This command is a wrapper for the API.

If an application creates different sessions for different purposes, it can be useful to create the sessions with names to identify them by, using the `XhiveDriverIf.createSession(String name)` method.

API documentation

com.xhive.core.interfaces.XhiveDriverIf

Managing Indexes

This chapter contains the following topics:

- **Indexes**
- **Index APIs and samples**
- **Path indexes**
- **Multipath indexes**
- **Multipath index merge**
- **Multipath indexing methods**
- **Value indexes**
- **Value indexing methods**
- **Full-text indexes**
- **Full-text indexing methods**
- **Metadata value indexes**
- **Metadata full text indexes**
- **Library indexes**
- **Library indexing methods**
- **ID attribute indexes**
- **ID attribute indexing methods**
- **Element name indexes**
- **Element name indexing methods**
- **Concurrent indexes**
- **Concurrent indexing methods**
- **Non-blocking incremental indexes**
- **Context conditioned indexes**
- **Context conditioned indexing methods**
- **Optimizing index performance**
- **Indexes and timezones**

Indexes

Various types of indexes can be used to speed up queries. Especially with large data sets, indexes are essential to query performance.

Indexes that are 'live' are updated automatically when the indexed data is updated. Since updating the data means updating the indexes, the number of indexes directly impacts update performance. The only non-live indexes are the [context conditioned indexes, page 168](#), which are deprecated, and supported only for the sake of compatibility with previous xDB versions.

The following live index types are currently supported:

- [path indexes, page 151](#)
- [full text indexes, page 163](#)
- [multipath indexes, page 153](#)
- [value indexes, page 161](#)
- [metadata indexes, page 164](#)
- [library ID indexes and library name indexes, page 165](#)
- [ID attribute indexes, page 166](#)
- [element name indexes, page 167](#)

Only libraries can have a library name index and/or a library ID index. The other index types can be defined for a library or for a document. Indexes are maintained automatically for all descendants and children of the library or document. The index is not locked with the library or document, to improve concurrent access to the index.

Most of the index types can be defined as either concurrent or compressed.

An index stores key-value pairs. The index key is a string, or in case of value indexes, a number type and a node set value. The index keys are always sorted. Indexes are scalable and the number of index keys and values can grow without limitations. The node sets can become large, especially with ID attribute, element name, and value indexes.

For information about index use with XQuery, refer to [Using indexes in XQuery, page 188](#).

Index APIs and samples

All index types use the **XhiveIndexIf** interface. For information about index use with versioned library children, refer to [Using searchable versions, page 125](#).

Samples

[FTI.java](#)

[CreateMultiPathIndex.java](#)

[LibraryIndexes.java](#)

[IdAttributeIndex.java](#)

[ValueIndex.java](#)

[ElementNameIndex.java](#)

[IndexAdder.java](#)

[IndexInConstruction.java](#)

API documentation

[com.xhive.index.interfaces.XhiveIndexIf](#)

[com.xhive.index.interfaces.XhiveIndexListIf](#)

[com.xhive.index.interfaces.XhiveIndexAdderIf](#)

[com.xhive.index.interfaces.XhiveSubPathIf](#)

[com.xhive.index.interfaces.XhiveExternalIndexConfigurationIf](#)

[com.xhive.index.interfaces.XhiveAnalyzer](#)

[com.xhive.index.interfaces.XhiveXqftOptionsAnalyzer](#)

[com.xhive.index.interfaces.XhiveIndexWithPathsAnalyzer](#)

[com.xhive.index.interfaces.XhiveIndexInConstructionIf](#)

[com.xhive.index.interfaces.XhiveIndexInConstructionListIf](#)

[com.xhive.index.interfaces](#)

Path indexes

Path indexes index the value of elements and attributes. Path indexes provide a more general way of specifying the indexed element and allow multiple element and attribute values to be used as index keys. For example, a full-text field index key can be used as one of the multiple values to accelerate `xhive:fts` queries.

Path indexes are specified using an XPath-like syntax. The syntax consists of a path to the indexed element and an optional specification of the values that are used as index keys. The element associated with the index key value can only contain a single text or CDATA child node.

Index option examples

```
//elem indexes all elements named elem. This is similar to an element value index. Such an index can speed up queries such as //elem[. = "value"], /foo/bar/elem[. = "value"], or /foo[bar/elem = "value"].
```

```
//elem[@attr] indexes all elem elements that contain attr attribute values. This index option is similar to an attribute value index and speeds up queries such as //elem[@attr = "value"].
```

```
//{http://www.example.com}
elem[@{http://www.example.com}attr] indexes all elem elements that contain attr attribute values that are in the http://www.example.com namespace.
```

```
//*[@attr] indexes all attr attribute values independent of the element name. This index option is similar to an attribute value index and speeds up queries such as //*[@attr = "value"] and //elem[@attr = "value"].
```

```
//chapter/title indexes all title elements that are nested in chapter elements. This index option can be used for queries such as //chapter/title[. = "Intro"].
```

`//chapter[title]` indexes all **chapter** elements that contain the specified **title** element value. This index option can be used for queries such as `//chapter[title="Intro"]`.

`/root[node/@id]` indexes the **id** attribute values of **node** elements nested in a **root** element. Using this type of path can speed up index updates because only the nested elements are searched and not the entire document.

`//elem[@attr1 + @attr2]` indexes all **elem** elements that contain **attr1** and **attr2** attribute values. This index option can be used for queries that require only a single lookup, such as `//elem[@attr1 = "value1" and @attr2 = "value2"]`, or a query such as `//elem[@attr1 = "value1" and @attr2]`.

`//elem<INT>` indexes the **elem** element values as integers, similar to a value index using the `TYPE_INT` option. This index option can be used for queries such as `//elem[. = 10]`. Specifying the type as an option is not possible in path indexes, because in a index with multiple values each value used as the index key can have a different type.

`//items/item[@id<INT> + price<FLOAT> + description/name<STRING>]` indexes values that have different types. This kind of index option can be use for queries such as `//items/item[@id = 10 and price = xs:float(4.53) and description/name = "keyboard"]`.

`//article[body<FULL_TEXT>]` indexes articles by the full-text content in the article body. This index option can be used for queries such as `//article[body contains text "apples"]`.

`//article[body<FULL_TEXT:my.package.CustomAnalyzer:>]` indexes articles by the full-text content in the article bodies using a custom text analyzer.

`//article[author<STRING> + body<FULL_TEXT::GET_ALL_TEXT, SA_ADJUST_TO_LOWERCASE, SA_FILTER_ENGLISH_STOP_WORDS>]` indexes articles on both author and content. This index option can be used for queries such as `//article[author="John" and body contains text "apples"]`.

`//elem[%{key1} + %{key2}]` indexes all **elem** elements where the ownerdocument contains metadata fields with name **key1** and **key2**. This index option can be used for queries that require only a single lookup, such as `//elem[xhive:metadata(., "key1") = "value1" and xhive:metadata(., "key2") = "value2"]`.

Path index specification

Path indexes use the following syntax:

spec	:=	elementpath (type '[' values ']')?
elementpath	:=	elementstep+
elementstep	:=	('/' '//') name
values	:=	value ('+' value)*
value	:=	('.' '@' name '%{' metadatakey '}' valuepath) type?
valuepath	:=	'./'? name (('/' '//') name)* ('/'@' name)?
type	:=	'<' ('STRING' 'INT' 'LONG' 'FLOAT' 'DOUBLE' 'DATE_TIME' 'DATE' 'TIME' 'YEAR_MONTH_DURATION' 'DAY_TIME_DURATION' fulltextspec) '>'

fulltextspec	:=	'FULL_TEXT' (?: analyzername? (?: fulltextoptions?)?)?
fulltextoptions	:=	(fulltextoption fulltextoption ', ' fulltextoptions)
fulltextoption	:=	('GET_ALL_TEXT' 'SUPPORT_PHRASES' 'SA_ADJUST_TO_LOWERCASE' 'SA_FILTER_ENGLISH_STOP_WORDS' 'INCLUDE_ATTRIBUTES')
name	:=	'*' uri? localname
uri	:=	'{ ([^&}] reference)* }'

In these specifications `metadatakey` is the metadata field name, `localname` is an XML name, `reference` is an XML character or predefined entity reference, and `analyzername` is a Java class name.

Some of the full-text options are only valid in combination with the `XhiveStandardAnalyzer`. For more information about the analyzer, see [Full-text indexes, page 163](#).

Multipath indexes

A multipath index allows you to index multiple elements without requiring explicit configuration of every single index path. It can index the contents of elements as specific value types or as full-text.

To specify which XML elements should be indexed, the user must specify the index' main path, and multiple sub-paths. All sub-paths are resolved relative to the main path, and are specified through an XPath-like path expression and a set of options. All elements matched by the path expression will be indexed using the given options.

The two most important configuration options for a sub-path are **VALUE_COMPARISON** and **FULL_TEXT_SEARCH**. They can be used together, allowing you to search the contents of an element either through value or through full-text search. The default type used for value comparison of a sub-path is `String`.

Lucene

The multipath indexes are implemented using Apache Lucene (<http://lucene.apache.org>), a high-performance, open-source search library. xDB stores multipath indexes separately in the database, as *lucene blobs*.

Internally, a multipath index consists of Lucene *segments*, small chunks that contain one or more indexed documents. New segments are added as new documents are being indexed.

Lucene segments are logically organized into so-called *sub-indexes* that represent updates to the index made by individual transactions. A search query into a multipath index corresponds to a query over the union of all applicable sub-indexes. For the sake of indexing and query performance, sub-indexes can periodically *merge*. For more information, refer to [Multipath index merge, page 156](#).

Multipath index examples

Assuming that the main path of the multipath index is `/main/path`, the table below shows various possibilities for specifying sub-path expressions.

Table 22 Sub-path path expression matching examples

Sub-path specification	Description
<code>/</code>	A single slash will match the root path of the index. In the example, this corresponds to <code>/main/path</code> . Assuming the sub-path option <code>FULL_TEXT_SEARCH</code> was used, the index will be used for queries such as <code>/main/path[. contains text "some text"]</code>
<code>* (or /*)</code>	A wildcard matching any direct child element of the main path of the index. Assuming subpath option <code>FULL_TEXT_SEARCH</code> , such a sub-path configuration will speed up queries such as <code>/main/path[elem contains text "some text"]</code> , or <code>/main/path[elem2 contains text "some text"]</code> .
<code>elem (or /elem)</code>	Indexes all elements named <code>elem</code> which are direct children of the index' main path, like elements with path <code>/main/path/elem</code> . If this sub-path uses the option <code>VALUE_COMPARISON</code> , it will be used in a query such as <code>/main/path[elem = "value"]</code> . If using the sub-path option <code>FULL_TEXT_INDEX</code> , it could be queried with <code>/main/path[elem contains text "some text"]</code> .
<code>elem/elem2</code>	This is a trivial extension of the previous example. Elements with path <code>/main/path/elem/elem2</code> will be indexed.
<code>{ns}elem</code>	Indexes all elements with the namespace URI <code>ns</code> and the local name <code>elem</code> which are direct children of the index' main path, so elements with path <code>/main/path/{ns}elem</code> .
<code>{ns}*</code>	Indexes all elements in the namespace URI <code>ns</code> which are direct children of the index' main path.
<code>{*}elem</code>	Indexes all <code>elem</code> elements in any namespace (including elements in no namespace) which are direct children of the index' main path.
<code>//elem</code>	This path will match any <code>elem</code> node. However, each distinct element path will be indexed separately. Assuming sub-path option <code>VALUE_COMPARISON</code> , the index will be used for queries like <code>/main/path[elem = "value"]</code> , or <code>/main/path[A/B/C/elem = "value"]</code> but not for a query such as <code>/main/path[//elem = "value"]</code> .
<code>//*</code>	This path will match any element. Therefore it can be used to index the contents of every single node. But in order to query it, the query must still specify the actual path to be queried. Note Combining this sub-path with the sub-path options <code>FULL_TEXT_INDEX</code> and <code>INCLUDE_DESCENDANTS</code> might create a very large index, depending on your XML document structure. It will cause all tokens from descendant elements to be included in parent elements. See also Multipath index limitations, page 160 .
<code>@attr (or /@attr)</code>	Indexes the attribute <code>attr</code> of the element corresponding to the index' main path, i.e. the attribute with the path <code>/main/path/@attr</code> . If this sub-path uses option <code>VALUE_COMPARISON</code> , it will be used in a query such as <code>/main/path[@attr = "value"]</code> . Using option <code>FULL_TEXT_INDEX</code> , it could be queried with <code>/main/path[@attr contains text "some text"]</code> .

Sub-path specification	Description
@* (or /@*)	A wildcard matching all attributes of the element corresponding to the main path of the index. Assuming sub-path option <code>FULL_TEXT_SEARCH</code> , such a sub-path configuration would allow the index to speed up queries such as <code>/main/path[@attr contains text "some text"]</code> , or <code>/main/path[@attr2 contains text "some text"]</code> .
elem/@attr	Indexes the attributes with the path <code>/main/path/elem/@attr</code> .
//elem/@attr	This path will match the attribute <code>attr</code> of any <code>elem</code> node. However, each distinct attribute path will be indexed separately. So, assuming sub-path option <code>VALUE_COMPARISON</code> , the index will be used for queries like <code>/main/path[elem/@attr = "value"]</code> , or <code>/main/path[A/B/C/elem/@attr = "value"]</code> but not for a query like <code>/main/path[//elem/@attr = "value"]</code> .
@{ns}attr	Indexes the attribute with the namespace URI <code>ns</code> and the local name <code>attr</code> of the element that corresponds to the index' main path, i.e the attribute with path <code>/main/path/@{ns}attr</code> .
@{ns}*	Indexes all attributes in the namespace URI <code>ns</code> of the elemt that corresponds to the index' main path.
@{*}attr	Indexes all <code>attr</code> attributes in any namespace (including elements in no namespace) of the element that corresponds to the index' main path.
//{ns}*/@attr	This path will match the attribute <code>attr</code> of any element in the namespace URI <code>ns</code> . Each distinct attribute path will be indexed separately.

Sub-path specification

Multipath indexes use the following syntax for sub-paths:

spec	:=	attr ((relpath abspath) ('/' attr)?)
relpath	:=	name elementstep*
abspath	:=	'/' elementstep+
elementstep	:=	('/' '//') name
attr	:=	'@' name
name	:=	(uri? wildcard) ((uri uriwildcard)? localname)
uri	:=	'{ ([^&}] reference)* }'
wildcard	:=	'*'
uriwildcard	:=	'{? wildcard }'

In these specifications `localname` is an XML name and `reference` is an XML character or predefined entity reference.

Differences between multipath and path indexes

Users familiar with path indexes may be confused by the similarities and differences between path indexes and multipath indexes. The table below describes differences between both index types.

Table 24 Differences between multipath indexes and path indexes

Aspect for comparison	Multipath index	Path index
Indexing multiple elements at the same time, in order to be able to query for all of these elements at the same time	Multipath indexes may have several different sub-paths indexed, and each sub-path is indexed independently of the others. So it is possible to query for each sub-path separately, or for all of them together.	Path indexes may index several different descendants of a base path through the syntax /main[a + b] but only documents with both a and b will be indexed. Therefore querying this index through /main[a = "value"] is not possible.
Indexing attributes of elements	Sub-paths may refer to attributes.	Path indexes can index attributes through the syntax /elem[@attributeName].
Full-text components	Multipath indexes do not impose limits on the number of sub-paths that use the option <code>SubPathOptions.FULL_TEXT_SEARCH</code> .	Path indexes can only have a single element being indexed as a full-text index. Therefore, you cannot create an index with path /path[elem<FULL_TEXT::> + elem2<FULL_TEXT::>].
Indexing elements as both by value and as tokenized full-text	Multipath indexes allow a sub-path to specify <code>SubPathOptions.VALUE_COMPARISON</code> together with <code>SubPathOptions.FULL_TEXT_SEARCH</code> , and in that case all elements matching the sub-path both by value as well as full-text will be indexed.	Path indexes also allow you to specify a node twice: once being indexed per value, and another as full-text, but as you have to search for these elements together, this is ineffective.

Multipath index merge

Over time, as update operations on the multipath index create new sub-indexes, the number of sub-indexes can become very large. As this can reduce indexing and query performance, smaller sub-indexes periodically *merge* into larger ones, keeping the number of sub-indexes at a minimum, and ideally producing a single, optimized index.

The merging policy is configurable. This requires care, because merging has direct impact on the overall performance of the system. While it is desirable to keep the number of sub-indexes small, sub-index merging can be time-consuming and CPU-intensive, and if done too often it can affect the performance of the page server. The following index merging tasks can be fine-tuned to achieve an optimal and balanced level of indexing and query performance:

Lucene internal segment merge	During indexing, Lucene segments are merged into larger ones as new data is being indexed.
Final merge	An asynchronous process merges sub-indexes into a single, optimized “final” index. A final merge is very I/O and CPU-intensive, and can take a long time to finish. Especially for large indexes, it should not be run during performance-critical periods.
Non-final merge	An asynchronous process merges smaller sub-indexes into larger ones. A non-final merge is more lightweight than a full final merge, so non-final merge can be executed more frequently.

The asynchronous merging tasks produce new sub-indexes as a result of merging smaller sub-indexes. The original sub-indexes will eventually be deleted by a periodic index cleaning task.

For information on configuration settings for merging and cleaning tasks, refer to [Multipath index properties, page 157](#).

Multipath index properties

Various aspects of multipath indexing can be configured using settings in the file [xdb.properties, page 66](#).

Note: For the settings to take effect, the file `xdb.properties` must be present in the page server’s Java classpath.

The settings are global for the page server, and apply to all multipath indexes in the federation. It is possible to override certain settings (such as `finalMergingInterval` and `finalMergeNoLogging`) for specific indexes using the API.

Table 25 Multipath index settings

Property	Description
<code>xhive.lucene.cleanMergeInterval</code>	The interval (in seconds) between the start of non-final merges. Default is 300.
<code>xhive.lucene.finalMergingInterval</code>	The interval (in seconds) between the start of final merges. Default is 14400.
<code>xhive.lucene.finalMergingBlackout</code>	The final merges blackout time window. This defines a daily period during which final merges are forbidden, using the form <code>StartHour-EndHour</code> (in 24h format). Default value is 0 (no blackout period). Example: 8-20 (blackout from 8AM to 8PM)
<code>xhive.lucene.cleaningInterval</code>	The interval (in seconds) between the start of periodic index cleaning tasks. Default is 120

Property	Description
xhive.lucene.blacklistsKeep	Indicates whether or not to keep blacklists. Default is <code>false</code> .
xhive.lucene.cleaningBlacklistInterval	The interval (in seconds) between the start of periodic cleaning of blacklists. Only used if <code>xdb.lucene.blacklistsKeep</code> is set to <code>false</code> . Default is 300.
xhive.lucene.refreshBlacklistCacheDuringMerge	Determines whether the blacklist cache will be refreshed during non-final merges. This may improve query performance under heavy ingest. Default is <code>false</code>
xhive.lucene.finalindex.size	The maximum number of sub-indexes in final-merged indexes. Default is 1.
xhive.lucene.nonFinalMaxMergeSize	The maximum size (in bytes) of a sub-index to be included in a non-final merge. Default is 300000000
xhive.lucene.mergeFactor	The number of documents in a segment that triggers the internal Lucene merge. Default is 10.
xhive.lucene.maxMergeDoc	The maximum number of documents in one segment. Default is 1000000
xhive.lucene.maxSegmentsForOptimization	The maximum number of segments in a sub-index. Default is 5.
xhive.lucene.parallelExecutionFinalMerge-CrossNodes	Indicates whether multiple final merges can be executed in parallel if there are multiple multipath indexes. Default is <code>true</code> .
xhive.lucene.mergingPolicyThreadPoolSize	The maximum number of threads to use for the asynchronous merging tasks. Default is 8
xhive.lucene.finalMergeNoLogging	Indicates whether transaction logging is disabled for final merge operations. Default is <code>true</code> . Note: Disabling of transaction logging for final merges significantly improves their performance, but it means that incremental backups will not include corresponding transaction log records for the multipath index. During incremental backup of a federation that includes a multipath index, either enable final merge transaction logging, or else exclude from the incremental backup all multipath indexes that have final merge transaction logging disabled. To back up such indexes, consider using standalone backup or library backup.
xhive.lucene.ramBufferSizeMB	The size (in MB) for the Lucene index writer <code>RAMDirectory</code> . Default is 3.
xhive.lucene.useCompoundFile	Indicates whether to use the Lucene compound file format. Default is <code>false</code> .
xhive.lucene.temp.path	The temporary directory for index entries. Default is blank (the system default temporary directory).

Property	Description
xhive.lucene.queryResultsWindowSize	The maximum number of documents returned by one query. Default is 12000.
xhive.lucene.ratioOfMaxDoc	The maximum term frequency (0-1) for terms to be accepted by wildcard queries. (Term frequency is calculated during indexing and stored with the Lucene index.) For example, if the frequency is 0.5 or higher, the queries will not accept terms that occur in more than half of the documents in the index, so that searching for <code>an.*</code> will be unlikely to return any hits for common words such as “and”. Default is 1.
xhive.lucene.termsExpandedNumber	The cutoff number of terms returned by wildcard queries. Default is 65536
xhive.lucene.fuzzyQueryPrefixLength	The number of leading characters to ignore when assessing similar terms in fuzzy queries. Default is 0.
xhive.lucene.fuzzyTermsExpandedNumber	The maximum number of similar terms to return by fuzzy queries. Default is 2147483647.
xhive.lucene.facetPathPatternMatch	For facet search, indicates whether to use the matching sub-path (<code>false</code>) or the XML node path (<code>true</code>) as the facet key. Default is <code>false</code> .
xhive.lucene.searchValueForFtcontains	Indicates whether sub-paths that specify the <code>VALUE_COMPARISON</code> option, but not the <code>FULL_TEXT_SEARCH</code> option, can be used when evaluating full-text search queries. Default is <code>false</code> .
xhive.lucene.ignoreValueComparisonScore	Indicates whether to ignore the score for value comparison queries on a multi-path index. Default is <code>true</code> .
xhive.lucene.strictIndexTypeCheck	Determines whether to perform strict type checking on indexed sub-paths. Default is <code>true</code> . If set to <code>false</code> , sub-paths with inconsistent value data types can still be indexed (as string). Note: This may cause non-conformant behavior with respect to the XQuery specification.
xhive.lucene.fieldCacheSize	The maximum number of entries in the query cache for Lucene fields and their corresponding sub-paths. Default is 4096.
xhive.lucene.maxBooleanClause	The maximum allowed number of Lucene BooleanClauses in a BooleanQuery. Default is 65536.

Multipath index limitations

Individual subpaths in multipath indexes that are configured as full text fields will only tokenize all text below the node if they are configured with the `INCLUDE_DESCENDANTS` option. This means that such a full text field is not usable for XQFT full-text queries of the form `/node[subpath contains text 'hello']`.

XQFT queries will only work together with sub-paths that either specify the `INCLUDE_DESCENDANTS` option or with sub-paths that index all fields below a path, e.g., `/**`.

Furthermore, sub-paths that do not have the `INCLUDE_DESCENDANTS` option enabled do not correctly support phrase queries if the phrase spans more than one XML element.

Given this XML fragment:

```
<doc>
  <field>
    <value1>Hello</value1>
    <value2>World</value2>
  </field>
</doc>
```

and a multipath index on `/doc` with a full text sub-path `/**` but no `INCLUDE_DESCENDANTS` option, phrase queries like `/doc[field contains text 'hello world']` will return no results, even though they should according to the XQFT specification.

A possible workaround is to enable the `INCLUDE_DESCENDANTS` option. However, for sub-paths like `/**` this may cause a significantly larger index, as every parent element will contain all tokens of its descendant elements.

Multipath indexing methods

While the main path is passed as an argument when creating a multipath index, all other options (including sub-paths) are specified through a configuration object. This is different from other xDB indexes, where general options are specified through an int bit mask. The configuration object must be of type `XhiveMultiPathIndexConfigurationIf`. It can be obtained through `XhiveIndexListIf.createExternalIndexConfiguration()`. Notice that this call will return an object of type `XhiveExternalIndexConfigurationIf`, which extends `XhiveMultiPathIndexConfigurationIf`.

Sub-paths are specified through a string representing a path expression used to match nodes for indexing, and a `XhiveSubPathIf` instance which carries the options for indexing the matched nodes.

The configuration option `VALUE_COMPARISON` is String by default, and can be changed using `XhiveSubPathIf.setType(int)`.

Samples

[CreateMultiPathIndex.java](#)

API documentation

[com.xhive.index.interfaces.XhiveIndexIf](#)

[com.xhive.index.interfaces.XhiveSubPathIf](#)

[com.xhive.index.interfaces.XhiveMultiPathIndexConfigurationIf](#)

[com.xhive.index.interfaces.XhiveExternalIndexConfigurationIf](#)

[com.xhive.index.interfaces.XhiveIndexWithPathsAnalyzer](#)

Customizing the score of multipath indexes

Multipath indexes allow customization of the score of the whole index through a user-defined class. Like the analyzer, this class, must be set in the configuration object before the creation of the index. This can be done through **XhiveMultiPathConfigurationIf.setScoreCustomizer(String)**.

The **XhiveScoreCustomizerIf** interface extends Lucene's **Similarity** methods by adding a **XhiveScoreCustomizerIf.getScoreBytes** method. The main difference with Lucene's **Similarity** is that the byte array, known as payload in Lucene, is not passed to index through the Analyzer but after the text has been tokenized.

If this interface is used, and if byte arrays have been stored for a given field, when the index is queried, the user-set implementation of **XhiveScoreCustomizerIf** will have access to this data, and will be able to customize the score of the result through its return value.

Samples

[CreateMultiPathIndex.java](#)

API documentation

[com.xhive.index.interfaces.XhiveScoreCustomizerIf](#)

[com.xhive.index.interfaces.XhiveMultiPathIndexConfigurationIf](#)

Value indexes

A value index stores elements by an element value or attribute value. Value indexes can be created for a library or a document. An index list can contain multiple value indexes.

Value indexes support namespaces. Use of namespaces requires the indexed documents to be parsed with the namespaces option enabled, and the value index to be created with Element URI or Attribute URI parameters.

xDB supports value indexing of:

- elements by element value.
- elements by attribute value.
- named elements by attribute value.

Using value types for value indexes

The following value types can be used to build a value index:

- TYPE_STRING - String value.

- TYPE_LONG - Long integer value.
- TYPE_INT - Integer value.
- TYPE_FLOAT - Single-precision floating point value.
- TYPE_DOUBLE - Double-precision floating point value.
- TYPE_DAY_TIME_DURATION - Integer values for day, time, and duration.
- TYPE_DATE_TIME - Integer values for day and time.
- TYPE_DATE - Integer value for date.
- TYPE_TIME - Integer value for time.
- TYPE_YEAR_MONTH_DURATION - Integer value for year, month, and duration.

By default, a value index contains string values.

All value types match XML Schema simple data types. The indexed documents do not necessarily have to comply with an XML schema that specifies the value types. As long as the values of all elements and attributes that are placed in the index match the value syntax, the index is constructed properly.

When selecting a type for an index, it is important to ensure that the indexed data adheres to the value type. If not, xDB throws an exception.

All data can be indexed using the string type. However, it can be useful to index data that includes decimal values using the double-precision floating point value. The sorting is based on the type rather than a lexicographical order. Different lexicographical representations of the same value are registered under the same index key.

When the XQuery processor uses an index to look for an integer value, it only looks for an integer value index. **Note:** XQuery uses value indexes, but they are not integrated within XPath and XPointer.

Value indexing methods

Value indexes are live indexes that are automatically updated when elements or attributes are inserted, replaced, or removed.

Value indexes support namespaces. Whenever namespaces are used, the indexed documents must have been parsed with the `XhiveIndexIf.PARSER_NAMESPACES_ENABLED` option and the `elementURI` or `attributeURI` parameters must be supplied to the `addValueIndex()` method.

The following example code creates a value index, using the `addValueIndex()` method. The parameters that are supplied to the `addValueIndex()` method determine the exact type of the value index.

```
// create a value index that stores elements by element value
XhiveIndexIf nameIndex =
    indexList.addValueIndex(nameIndexName, null, "NAME", null, null);

// create a value index that stores elements by attribute value
XhiveIndexIf IDIndex = indexList.addValueIndex(idIndexName, null, null, null, "ID");

// create a value index that stores named elements by attribute value
XhiveIndexIf personByFatherIndex =
    indexList.addValueIndex(personByFatherIndexName, null, "PERSON", null, "FATHER");
```

Samples

[ValueIndexIndex.java](#)

API documentation

[com.xhive.index.interfaces.XhiveIndexListf](#)

Full-text indexes

A full-text index is a special form of value index that stores elements by element text values or an attribute value. Full-text indexes are more versatile but slower than value indexes, especially during updates. Full-text indexes support namespaces. Use of namespaces requires the indexed documents to be parsed with the namespaces option enabled, and the full-text index to be created with Element URI or Attribute URI parameters.

An index list can contain multiple full text indexes. Full-text indexes can be used to:

- Search for individual words of an element value.
- Perform complex Boolean and wildcard queries.
- Index all the underlying text of an element and subelements.

There are some index options specifically for full-text indexes:

Table 26 Full-text index options

Index option	Description
FTI_GET_ALL_TEXT	The element is indexed by its string value, which is computed from the string value of all descendant nodes. If this option is not enabled, the element can only have text-child nodes, but the index updates faster.
FTI_SUPPORT_PHRASES	Optimizes the index to perform phrase queries. Using this option increases the index size.
FTI_SUPPORT_SCORING	Stores extra information about the indexed tokens to improve the score calculation quality.
FTI_SA_ADJUST_TO_LOWERCASE	Converts the indexed terms to lower case to support queries that are not case-sensitive.
FTI_SA_FILTER_ENGLISH_STOP_WORDS	Applies a stop word filter. Words on the stop word list are not indexed.
FTI_INCLUDE_ATTRIBUTES	Indexes words that are part of the element attribute values. This option has no effect on full-text indexes placed on attributes. This option should be used in conjunction with the include-attrs option and the xhive:fts function.
FTI_LEADING_WILDCARD_SEARCH	Enables to search the index for terms with a leading wildcard. This option improves the speed of "PREFIX*SUFFIX" type searches. This option does not slow down searches without wildcards, but may increase the time it takes to update the index.

Full-text indexing methods

Specifying a custom analyzer class allows finer control over the full-text indexing process. The analyzer must be a `org.apache.lucene.analysis.Analyzer` subclass to act as a tokenizer. For more information about specifying an analyzer, see [Using the `xhive:fts` function, page 201](#).

Whenever namespaces are used, the indexed documents must have been parsed using the `XhiveIndexIf.PARSER_NAMESPACES_ENABLED` option, and the `elementURI` or `attributeURI` parameters must be supplied to the `addFullTextIndex()` method.

Example

The following example code uses the `addFullTextIndex()` method to create a full-text index. The parameters that are supplied to the `addFullTextIndex()` method determine the exact value index type.

```
// create a full text index on the text-contents of the name elements
XhiveIndexIf nameIndex = indexList.addFullTextIndex(nameIndexName,
    null, "NAME", null, null, null,
    XhiveIndexIf.FTI_SUPPORT_PHRASES | XhiveIndexIf.FTI_GET_ALL_TEXT);
```

Samples

[IndexAdder.java](#)

API documentation

[com.xhive.index.interfaces.XhiveIndexIf](#)

Metadata value indexes

Metadata value indexes are like value indexes, but instead of indexing the content of a node, they index the value of a metadata entry. The index key is the value of the metadata key, and index entries point to individual documents.

Metadata indexes are used for XQueries of the form:

```
doc('...')[xhive:metadata(., 'key') = 'value']
```

Metadata full text indexes

Metadata full text indexes are like metadata value indexes, but instead of indexing the metadata value, they create a full text index of the value of a metadata entry. The index key is the value of the metadata key, and index entries point to individual documents.

Metadata full text indexes are used for XQueries of the form:

```
doc('...')[xhive:metadata(., 'key') contains text 'value']
```

.

Library indexes

A library index indexes the content of a library. Library indexes must have a unique name.

There are two types of library indexes:

- Library ID index: uses the IDs of the library content objects, such as documents, libraries, and BLOBs. Library ID indexing is efficient when many content objects are stored in the library.
- Library name index: uses the names of the library content objects, and improves performance of XLink operations and full path XPointer queries.

By default, a library has a library name index. The root library also has a library ID index. A library can only have one library ID index and one library name index. Adding a second library ID or library name index generates an exception.

Names are not mandatory for library content, and some content is not included in the index.

Library indexing methods

A library index indexes the content of a library. Library indexes must have a unique name and are stored in an XhiveIndexListIf list index. The size of an existing index can be found through [XhiveIndexIf.getStoragePages\(\)](#).

Creating a library ID index

A library ID index improves the performance and scalability of the `get(long Id)` method in XhiveLibraryIf interface. Library ID indexing is efficient when many content objects are stored in the library.

The following code sample shows how to add a library ID index:

```
//get the index list of the library
XhiveIndexListIf indexList = library.getIndexList();

//add a library id index to the library
String idIndexName = "Library ID Index";
XhiveIndexIf idIndex = indexList.addLibraryIdIndex(idIndexName);
```

Creating a library name index

A library name index improves the performance and scalability of the `get(String name)` method in the XhiveLibraryIf interface, and improves performance of XLink operations and full path XPointer queries. Names are not mandatory for library content and some content is not included in the index.

The following code sample shows how to add a library name index:

```
//add a library name index to the library
String nameIndexName = "Library Name Index";
XhiveIndexIf nameIndex = indexList.addLibraryNameIndex(nameIndexName);
```

ID attribute indexes

An ID attribute index stores elements by their unique element ID. Users typically do not access ID attribute indexes directly, they are used implicitly to improve the performance of element ID operations and XQuery/XPath/XPointer queries.

ID attributes can be specified in the DTD or XML-schema associated with a document.

ID attribute indexes can be created for a library or a document. Element IDs are only unique within the context of a document. Since a library can contain more than one document, each document could use the same element IDs. The index does not limit the number of entries for a given key.

ID attribute indexing methods

Typically, ID attribute indexes are used implicitly to improve the performance of the `getElementById(String elementId)` DOM method and XQuery/XPath/XPointer queries.

ID attributes can be specified in the DTD or XML-schema associated with a document. Attributes that are created using the `createAttribute(String name)` DOM call are converted to ID attributes if they are defined in the DTD associated with the document. Instead of using a DTD, ID attributes can be created using the `createIDAttribute(String name)` method in the `XhiveDocumentIf` interface.

The indexes use the `XhiveIndexIf` class and are stored in an `XhiveIndexListIf` index list. Index lists can store indexes of different types. Each index in the index list must have a unique name and an index list can only contain one ID attribute index.

Examples

The following code sample shows how to add an ID attribute index to a document.

```
//Get the indexlist
XhiveIndexListIf indexList = document.getIndexList();

//add the id attribute index to the indexlist of the document if the index is not found
String indexName = "ID Attribute Index";
XhiveIndexIf index = indexList.getIndex(indexName);
if (index == null){
    index = indexList.addIdAttributeIndex(indexName);
}
```

XQuery, XPath, and XPointer queries, and the `getElementsById(String elementId)` method use ID attribute indexes. Users typically do not access this type of index directly. However, the following code sample shows how to view the keys of an ID attribute index:

```
//Print the element of key = "p3"
String key = "p3";

XhiveNodeIteratorIf nodeIter = index.getNodesByKey(key);
if (nodeIter.hasNext()){
    System.out.println(" Element = " + nodeIter.next());
}
```

Element name indexes

An element name index stores elements by name. Element name indexes can be created for a library or document.

xDB supports two types of element name indexes:

- Element name index - All element names are indexed.
- Selected element name index - Only a selection of element names is indexed.

The selected element name index can be updated much faster than the element name index.

Element names in the selected element name index can be specified in one of the following ways:

- String *nodeName* , when the namespaceURI of the element is null.
- String *namespaceURI* + ' ' + *localName* , when the namespaceURI of the element is not null.
- String '{' + *namespaceURI* + '}' + *localName* , when the namespaceURI of the element is not null.

The keys used for indexing the elements are strings that contain both namespaceURI and local name, separated by one single space character. For example, an element with the namespaceURI **http://www.x-hive.com** and the local name **chapter** is indexed with the key **http://www.x-hive.com chapter**.

Element name indexing methods

Example

The following code sample shows how to create a selected element name index.

```
//Add a selected element name index
String[] names = {"NAME", "BORN", "WIFE"};
XhiveIndexIf selectedElementNameIndex = indexList.getIndex(selectedElementIndexName);
if (selectedElementNameIndex == null){
    indexList.addElementNameIndex(selectedElementIndexName, names);
}
```

For namespaces, the selected element names can be defined as follows:

```
//Define the names of an element name index with namespaces
String[] names = {"http://www.x-hive.com chapter", "{http://www.x-hive.com}owner"};
```

Concurrent indexes

Concurrent indexes are not locked for the duration of the transaction when accessed or modified. Only the used pages are latched, and only for the time that they are read or modified. This process improves concurrency at the expense of some extra overhead when using the indexes. Whether the net effect is beneficial depends on your application.

Concurrent indexing methods

To specify an index as concurrent, use the `XhiveIndexIf.CONCURRENT` flag when creating the index.

The following restrictions apply to concurrent indexes:

- When using the `XhiveIndexIf.getKeys()` method outside of `XQuery`, the index can return keys that have no nodes.
- Concurrent indexes do not provide phantom protection. If the index is read a second time during the same transaction, new keys and nodes that have been committed since the previous lookup can appear. Keys and nodes read can never disappear, because the actual data read is still read locked for the duration of the transaction.
- Concurrent indexes do not have a separate authority value. The `XhiveIndexIf.getAuthority()` method returns the authority of the owning library child.

Non-blocking incremental indexes

The regular indexes acquire a write lock on a library until the building process, which runs in a single transaction, completes. If you need to build or rebuild an index on a large library without blocking updates to it, a special implementation of an index should be used: **`XhiveIndexInConstructionIf`**. It allows the index to be built on demand in as many separate transactions as you need and in batches of any size. It supports concurrent indexing sessions. These indexes are created in a dedicated **`XhiveIndexInConstructionListIf`** and, after the building process completes, you move them to the ordinary index list to be used by queries.

The supported indexes are: `PathValue`, `MultiPath`, `Value`, `Metadata`, `IdAttribute` and `ElementName`.

Note: This feature does not support versioned documents.

Sample

[IndexInConstruction.java](#)

API documentation

[com.xhive.index.interfaces.XhiveIndexInConstructionIf](#)

[com.xhive.index.interfaces.XhiveIndexInConstructionListIf](#)

Context conditioned indexes

Context conditioned indexes are **deprecated**.

A context conditioned index stores node objects by a user-defined key. A node can be an element, text node, processing instruction, comment, or document. Each context conditioned index has a user-defined index node filter. The filter is used to determine which nodes are included in the index and what key is used.

Context conditioned indexes are non-live indexes that are not automatically updated. An application program must update the indexes periodically.

Context conditioned indexes are used for:

- Searching for elements by element value or attribute value when the search cannot use value indexes.
- Complex queries.

In certain cases, a context conditioned index is faster than the equivalent XQuery, XPath, or XPointer query.

Context conditioned indexing methods

Note: Context Conditioned Indexes are deprecated as of xDB version 10.0.

To create a context conditioned index:

1. Get a handle to the index list, similar to the following:

```
// get the index list that belongs to this database
XhiveIndexListIf indexList = charterLib.getIndexList();
```

2. Create an **XhiveCCIndexIf** object, similar to the following:

```
// create an XhiveIndexIf object
String indexName = "Index of even numbered chapters";
XhiveCCIndexIf index = (XhiveCCIndexIf) indexList.getIndex(indexName);

if ( index != null ) {
    // remove existing index first
    indexList.removeIndex(index);
}
```

3. Create an index node filter using the **XhiveIndexNodeFilterIf** interface to define which nodes to include in the index, similar to the following:

```
index = (XhiveCCIndexIf) indexList.addNodeFilterIndex
("samples.manual.SampleIndexFilter", indexName);
```

4. Add context conditioned index entries for a document using the `indexDocument()` method in the **XhiveCCIndexIf** interface, similar to the following:

```
index.indexDocument(newDocument);
```

Example

The code sample below uses the created index to retrieve all titles of even number chapters:

```
Iterator keyIter = index.getKeys();
while (keyIter.hasNext()) {
    String key = (String) keyIter.next();
    System.out.println(key);
}
```

The following example code uses the `getNodesByKey()` method to retrieve nodes from the index based on a given key:

```
XhiveNodeIteratorIf nodesFound = index.getNodesByKey("AMENDMENTS");
```

```
while (nodesFound.hasNext()) {
    XhiveNodeIf docFound = (XhiveNodeIf)nodesFound.next();
    System.out.println(docFound.toXml());
}
```

Optimizing index performance

To achieve the best index performance:

- Do not add more indexes than required.
- Reduce [index scope](#).
- Make indexes as [selective](#) as possible. For example, use selected element name indexes instead of default element name indexes.
- Create indexes after loading the data. Indexing existing data is faster overall than updating indexes each time a new data item is added.

Index scope

xDB supports nested library structures. Users are free to select the scope context of an index. For example, the scope of the root library is larger than the scope of a nested library. The smaller the index scope, the better the performance of data updates and queries can be.

Note: The scope of library indexes is of no importance because library indexes only apply to the direct children of the library.

Index selectivity

In general, indexes provide the best query and update performance when the index is as selective as possible. Each key must have the smallest possible number of nodes.

Library indexes have both unique names and IDs, provide optimal selectivity, and therefore have excellent query and update behavior. Value indexes and ID attribute indexes are the next best choice for selective indexes. Element name indexes are not selective because most documents do not have unique element names. To maintain a good data update performance, it is better to use selected element name indexes instead of the default element name indexes. Selected element names only index a subset of all elements.

Ignoring indexes

Queries can disable certain indexes by providing a comma-separated list of index names with the `xhive:ignore-indexes` option. These indexes are not used to optimize the associated query.

```
declare option xhive:ignore-indexes 'myindex1,ftsindex';
for $x in ...
```

Indexes and timezones

The following indexes support date-time data types that can contain timezone information like `xs:date`, `xs:dateTime` and `xs:time`:

- [Value Index, page 161](#)
- [Path Index, page 151](#)
- [Mutipath Index, page 153](#)

To enable fast lookups, indexes have to store the index keys by order. It is not possible to mix date-time keys with and without timezone information, because the ordering of the keys would then depend on the implicit timezone of a query. For more information on implicit timezones, see [xhive:implicit-timezone](#).

To ensure non-ambiguous ordering, xDB now stores date-time keys with timezone PT0H. This is achieved as follows:

- Key values with timezone information are normalized to PT0H before they are stored in the index.
- Key values without timezone information are assigned to timezone PT0H.

Note: Because all stored date-time keys have timezone PT0H, this may lead to unexpected query results in older applications that store date-times without timezone information.

For example, consider using a Path index with path:

```
//doc[creationDate<DATE_TIME>]
```

and query:

```
let $dt := xs:dateTime("2007-09-14T08:00:00")
return //doc[creationDate = $dt]
```

If the query does not use the index then the result of the comparison between `creationDate` and `$dt` is not depending on the implicit timezone of the query. However, when the index is used, the query compares date-times with timezone information (`creationDate`) to a date-time without timezone information (`$dt`). According to the [XPath and XQuery Functions and Operators 3.0, fn:dateTime-equal](#), the date-time without timezone information must be adjusted to the implicit timezone before comparison. The result of this comparison now depends on the implicit timezone, even if the stored date-times do not depend on timezone information.

Note: To avoid unexpected timezone adjustments, it is now good practice to always set the implicit timezone to PT0H. Existing applications which currently use date-times without timezone should be modified accordingly.

XQuery

This chapter contains the following topics:

- [Working with XQueries](#)
- [Working with XQuery methods](#)
- [External XQuery variables and functions](#)
- [Accessing documents and libraries with XQuery](#)
- [XQuery error reporting](#)
- [XQuery options and extension expressions](#)
- [XQuery extension functions](#)
- [Using indexes in XQuery](#)
- [Proprietary XQuery extension to order by](#)
- [Using type information in XQuery](#)
- [XQuery full-text search](#)
- [Using the xhive:fts full-text search function](#)
- [XQuery performance tuning](#)
- [XQuery collation support](#)
- [XQuery Profiler](#)
- [XQuery profiling methods](#)
- [XQuery implementation](#)

Working with XQueries

This chapter discusses the use of XQuery in xDB applications, describes XQuery-related features and utilities, and gives details on xDB's implementation of XQuery.

Working with XQuery methods

To execute XQuery queries in xDB, you can use the `executeXQuery(String query)` method on the `XhiveNodeIf` interface. It returns an `XhiveXQueryResultIf` that is an iterator over the result sequence. Each element of the result is an instance of the `XhiveXQueryValueIf` object. For example:

```
XhiveNodeIf lc = ... ;  
XhiveXQueryResultIf result = lc.executeXQuery("doc('doc')//item");
```

```
while (result.hasNext()) {
    XhiveXQueryValueIf value = result.next();
    // We know this query will only return nodes.
    Node node = value.asNode();
    // Do something with the node ...
}
```

Within the query, the context item (accessible via the `.` operator) is initially bound to the node on which the query was executed:

```
XhiveNodeIf node = ...;
XhiveXQueryResultIf result = node.executeXQuery("./author/first, ./author/last,
                                                ./contents");

// using a Java 5 for each loop
for (XhiveXQueryValueIf value : result) {
    // do something with the value ...
}
```

If you only want to display the result, you can use the **toString()** method on the values returned, regardless of their type:

```
XhiveLibraryChildIf lc = ... ;
String query = ... ;
XhiveXQueryResultIf result = lc.executeXQuery(query);
while (result.hasNext()) {
    System.out.println(result.next().toString());
}
```

For more control over serialization, nodes can be serialized using an **LSSerializer** obtained from a library using **XhiveLibraryIf.createLSSerializer()**.

If the query uses node constructors, nodes are created in a temporary document. If desired, these nodes can be inserted into another document using the DOM **importNode()** method. If you want to insert the nodes into a particular document, specifying an owner document for new nodes in the call is more efficient than creating a temporary document and importing its nodes into the destination document. For example:

```
XhiveLibraryChildIf lc = ... ;
XhiveDocumentIf doc = ... ; // Create new nodes in this document
XhiveXQueryResultIf result = lc.executeXQuery("<count>{count(//item)}</count>", doc);
// We know this query will only return a single node.
XhiveXQueryValueIf value = result.next();
Node node = value.asNode();
// Append it to the document element of destination document
doc.getDocumentElement().appendChild(node);
```

The query result is evaluated lazily each time the **next()** method is called on the result iterator. Avoid calling **result.next()** within the same session after modification of searched documents or libraries, as undefined results may occur. If you want to use the query output to modify the searched documents, use [extension function **xhive:force\(\)**](#) or the [update syntax, page 211](#).

API documentation

com.xhive.dom.interfaces.XhiveNodeIf

External XQuery variables and functions

In xDB, declaring external variables and functions is not strictly necessary, but it is recommended for compatibility with other XQuery implementations.

External variables

XQuery provides a way to import external values into the query scope (parameters). To use this feature in xDB, create a query using the **createXQuery(String query)** method on a **XhiveNodeIf** interface. This method parses the query, resolves module imports, and returns an **XhiveXQueryQueryIf** object that represents the query.

Note: The **XhiveXQueryQueryIf** object is only valid for the current database session, so do not try to use it across multiple sessions.

Example

```
XhiveNodeIf node = ...;
XhiveXQueryQueryIf query = node.createXQuery(
    "declare variable $pi external; " +
    "for $rad in doc('radius.xml')//radius " +
    "return ($rad * $rad * $pi)");
query.setVariable("pi", java.lang.Math.PI);
Iterator<XhiveXQueryValueIf> result = query.execute();
...
```

XQFT Example

The syntax for using variables in **XQFT (XQuery Full-Text)** requires braces around the variable:

```
XhiveNodeIf node = ...;
XhiveXQueryQueryIf query = node.createXQuery(
    "declare variable $term external; " +
    "/book/chapter[. contains text {$term} ]/title");
query.setVariable("term", "bicycle");
Iterator<XhiveXQueryValueIf> result = query.execute();
...
```

The **XhiveXQueryQueryIf** interface also provides a **executeOn(XhiveNodeIf node)** method, that allows you to run the same query multiple times on different context items. When using **executeOn()**, the initial context item points to the given **XhiveNodeIf** interface.

The **XhiveXQueryQueryIf** interface automatically maps the following Java types to their corresponding XML Schema types.

Table 27 Java type mapping

Java object type	XQuery value type
boolean / java.lang.Boolean	xs:boolean
java.lang.String	xs:string
int / java.lang.Integer	xs:int
long / java.lang.Long	xs:long
java.math.BigInteger	xs:integer
float / java.lang.Float	xs:float
double / java.lang.Double	xs:double
java.math.BigDecimal	xs:decimal
javax.xml.namespace.QName	xs:QName
javax.xml.datatype.XMLGregorianCalendar	xs:dateTime and subtypes, depending on actual schema type
javax.xml.datatype.Duration	xs:duration and subtypes, depending on actual schema type
java.sql.Time	xs:time
java.sql.Timestamp	xs:dateTime
java.sql.Date	xs:date
java.util.Date	xs:dateTime
java.util.Calendar	xs:dateTime
org.w3c.dom.Node	A node of the corresponding type, such as an element, document, or similar.

All built-in DOM objects can be used directly, including the `XhiveLibraryIf` object. Nodes from another DOM implementation are imported into the creation document for this XQuery. For more information, see the `XhiveXQueryQueryIf.setCreationDocument` method.

Values that cannot be mapped are converted into a special Java value. For more information, refer to [Java objects and instance methods, page 218](#).

It is also possible to supply an `Iterator` over a sequence of objects. This can be especially handy for executing XQuery queries over the results of other queries, effectively creating a lazily executed XQuery pipeline. Iterators used by a query cannot be reused afterwards, not even by the same query. The declared variable is empty if this query is run again.

Example

```
XhiveNodeIf node = ...;
Iterator<XhiveXQueryValueIf> subresult = node.executeXQuery(...);
XhiveXQueryQueryIf query = node.createXQuery(
    "declare variable $values external; " +
    "for $value in $values " +
    "return $value + 5");
query.setVariable("values", subresult);
```



```
Iterator<XhiveXQueryValueIf> result = query.execute();
...
```

Custom functions

Custom functions can be set using the `setFunction(String, XhiveXQueryExtensionFunctionIf)` method. For more information, see the [XhiveXQueryExtensionFunctionIf API documentation](#).

Due to optimizations, your function may be called at a different moment during evaluation than you may expect. Therefore, it is best to avoid side effects in your extension functions.

Example

```
XhiveNodeIf node = ...;
XhiveXQueryQueryIf query = node.createXQuery(
    "declare function circle-area($radius as xs:number) as xs:double external;" +
    "for $rad in doc('radius.xml')//radius " +
    "return circle-area($rad)");
query.setFunction("circle-area", new XhiveXQueryExtensionFunctionIf() {
    public Object[] call(Iterator<? extends XhiveXQueryValueIf>[] args) {
        double rad = args[0].next().asDouble();
        double res = java.lang.Math.PI * rad * rad;
        return new Object[]{ new Double(res) };
    }
});
Iterator<XhiveXQueryValueIf> result = query.execute();
...
```

All methods for setting variables and functions can either:

- take a single `string` argument as the name and place the variable or function in the empty namespace, or
- take two `string` arguments: a namespace URI and a local name.

Using a namespace URI requires declaring a prefix within the query.

API documentation

[com.xhive.dom.interfaces.XhiveNodeIf#createXQuery\(java.lang.String\)](#)

[com.xhive.dom.interfaces.XhiveXQueryExtensionFunctionIf](#)

Accessing documents and libraries with XQuery

Queries can refer to specific documents using the XQuery `doc()` function. The function argument is a path, optionally starting with a `/` and containing names or IDs of libraries or documents, separated by `/`. Paths that do not start with a `/` are evaluated relative to the context item or the parent library on which the query is executed. If the path designates a library, the function returns a sequence of all documents in that library and its descendant libraries.

Examples

```
doc("/"),           (: All documents in the database :)
doc("/document.xml"), (: Document "document.xml" in the root library :)
doc("/MyLibrary"), (: All documents in "MyLibrary" :)
doc("/id:10"),      (: The document(s) in the library with id "10" :)
doc("/mylib/mydoc"),
doc("/mylib/mysublib/id:1234")
doc("relative/path")
doc("../steps/work/./too")
```

The argument does not have to be a string literal but can be any expression returning a string.

If there is a context node, an absolute path expression starts at the root of the `fn:root(.)` context node. In an outer expression it starts at the document or all documents in the library on which the XQuery was executed, or the document containing an initial context node. You can use `xhive:input()` to access the calling documents when there is a context node.

```
/docelem[//@id="2"]
(: this is equivalent to :)
xhive:input()/docelem[root(.)//@id="2"]
```

xDB also resolves URLs passed to the `doc()` function, like `doc('http://example.com/mydoc.xml')` by retrieving the document and parsing it. The URL is resolved using Java's `java.net.URL` class, so all URI schemes supported by Java are available from XQuery. **Note:** Applications can control this behaviour by means of an XQuery security policy. Applications can control document resolution in XQuery through a custom XQuery resolver.

Note: In xDB, the `collection()` function is similar to the `doc()` function, except that `collection()` can be called without any parameter.

XQuery error reporting

Errors within XQuery processing are reported by throwing exceptions extending the `XhiveXQueryException` class, which in turn extends the `XhiveException` class. The `com.xhive.error.xquery` package contains the XQuery-related exceptions:

Table 28 XQuery exceptions

Exception	Thrown on
<code>XhiveXQueryErrorException</code>	Semantic errors within the query. Thrown either directly or on one of its subclasses: <code>XhiveXQueryTypeException</code> or <code>XhiveStackOverflowException</code> .

Exception	Thrown on
XhiveXQueryTypeException	Errors related to the type system, for example when a supplied value did not match the expected type. This exception is a subclass of the XhiveXQueryErrorException class.
XhiveStackOverflowException	Stack overflows in user defined functions. This exception is a subclass of the XhiveXQueryErrorException class.
XhiveXQueryParseException	Parse errors in the query.
XhiveXQueryFTSParseException	Incorrect FTS query.
XhiveXQueryUnknownFunctionException	Use of an unknown function in a query.
XhiveXQueryUnsupportedException	Use of an unsupported feature, or declared XQuery version is greater than 1.0.
XhiveXQueryInternalException	Internal query errors.

XQuery options and extension expressions

xDB supports a number of [XQuery options, page 179](#). Those options can be set globally, or for a specific part of a query.

- To set an option globally, you can use the following syntax in the query prologue: `declare option QName "Value";`
Quotes around the `Value` parameter are required.
- To set an option for a specific part of a query, you can use an extension expression. Extension expressions are specified using the following syntax: `(# QName Value #)`
`{ expr }`
The `QName` option is set for the entire inner expression. Quotes around the `Value` parameter are optional. Multiple options can be set at once by writing multiple `(# #)` parts before the curly braces part.

Table 29 XQuery options

Option	Description
xhive:index-debug	Checks if an index is used in a query. When its value is different from the empty string, the query evaluator produces a message whenever a value is looked up in an index selected by the optimizer.
xhive:queryplan-debug	Similar to xhive:index-debug, but shows how the query is divided into parts, the order in which the parts are executed, and which indexes and options are looked up.
xhive:pathexpr-debug	Similar to xhive:index-debug, but shows which low level expressions within the XQuery are executed, and in what order.

Option	Description
xhive:optimizer-debug	Similar to xhive:queryplan-debug, but shows how the query optimizer tries to create an index plan for a path expression. The output contains detailed information about the indexes that are considered, including those that are eliminated, and how a query plan is constructed. The contents of the output are currently not documented.
xhive:ignore-indexes	Provides a comma-separated list of indexes that should not be used to optimize accesses.
xhive:index-paths-values	Provides a comma-separated list of paths whose values to retrieve directly from a multipath index.
xhive:fts-analyzer-class	Configures a fully specified analyzer classname to use in text searches for a full-text query or the xhive:fts function. If an index is present, the value of this option takes precedence over the analyzer.
xhive:fts-implicit-conjunction	Specifies the implicit conjunction operator for full-text searches. The only valid values are AND and OR . Default is OR .
xhive:fts-similarity-class	Configures a fully specified classname of the similarity that is used for score calculation in the full-text query.
xhive:fts-thesaurus-class	Configures a fully specified classname of the thesaurus handler used in the full-text query. If a thesaurus handler is already set by API, the option takes precedence.
xhive:timer	Specifies a timer for the encapsulated expression.
xhive:max-tail-recursion-depth	Specifies the maximum recursion depth for tail recursive functions. Default is 10000.
xhive:implicit-timezone	Specifies the implicit time zone, used by functions and operations in the various date types, such as xs:date, if no explicit time zone is supplied. The default implicit time zone is PT0H. When set to an empty string, the local time zone is used. For more information refer to Indexes and timezones, page 171 .
xhive:return-blobs	If set to true, changes the behaviour of the doc() and related functions to return also BLOBs.
xhive:return-versions-all	Changes the behaviour of the doc() and related functions to return all versions of queryable documents. Other documents, such as versioned documents that have not been created with the queryable parameter set to true, will behave as if the option is not defined (meaning that they always only return the latest version).
xhive:return-versions-at-date	Changes the behaviour of the doc() and related functions to return the last version of queryable documents that has a timestamp less than the specified timestamp. The timestamp is defined as an xs:dateTime string (see example below). For regular documents the same caveat applies as with xhive:return-versions-all.

Option	Description
xhive:return-versions-before-date	Changes the behaviour of the doc() and related functions to return all versions of queryable documents that existed before the specified timestamp. Can be used in conjunction with xhive:return-versions-after-date to define a closed date range. For regular documents the same caveat applies as with xhive:return-versions-all.
xhive:return-versions-after-date	Changes the behaviour of the doc() and related functions to return all versions of queryable documents that existed after the specified timestamp. Can be used in conjunction with xhive:return-versions-before-date to define a closed date range. For regular documents the same caveat applies as with xhive:return-versions-all.

Examples

```

declare option xhive:index-debug "true";
doc("/products")//product[@product_id = "42"]

(# xhive:index-debug "true" #) {
  doc("/products")//product[@product_id = "42"]
}

(# xhive:queryplan-debug "true" #)
(# xhive:pathexpr-debug "true" #) {
  doc("/products")//product[@product_id = "42"]
}

(# xhive:queryplan-debug "true" #)
(# xhive:optimizer-debug "true" #) {
  doc("/products")//product[@product_id = "42"]
}

declare option implicit-timezone 'PT10H';
adjust-dateTime-to-timezone(xs:dateTime("2002-03-07T10:00:00-07:00"))

(# xhive:fts-implicit-conjunction 'AND' #) {
document("/manual")//paragraph[xhive:fts(., "long list of words")] /text()
}

declare option xhive:return-versions-at-date '2001-01-01T00:00:00Z';
doc("/products")//product[@product_id = "42"]

```

XQuery extension functions

xDB implements some useful functions that are not part of the XQuery Working Draft. These extension functions are all in namespace `http://www.x-hive.com/2001/08/xquery-functions` which is bound to prefix `xhive` by default.

- **xhive:fts(\$context as node(), \$query as xs:string, \$options as xs:string) as xs:boolean**

The function **xhive:fts** executes a query using xDB full-text index. The `$options` argument is optional, and should be a string literal containing a semicolon-separated list of options. The `include-attrs` option executes a query using a full-text index, but also looks in attributes of elements under `$context`. For more information, refer to the section on [using the xhive:fts full-text search function, page 201](#).

Example:

```
doc("/library")//chapter[xhive:fts(title, "venice and merchant*")]
```

- **xhive:evaluate(\$query as xs:string, [\$var1 as xs:QName, \$val1 as item()*, \$var2 as xs:QName, \$val2 as item()*, ...]) as item()***

This function evaluates a single string argument as an XQuery query and returns the result of the query. For example:

```
for $query in doc("/queries")//query
return
  <queryresult>
    { $query }
    <result>{ xhive:evaluate($query) }</result>
</queryresult>
```

The values for external query variables can be specified using additional arguments to the `xhive:evaluate()` function. The number of these arguments must be even, and they must alternate between QNames (name of the external variable) and items (value of the external variable).

- **xhive:parse(\$doc-text as xs:string, \$schema-hint as xs:string) as document-node()**

xhive:parse(\$doc-text as xs:string) as document-node()

These functions take the serialized text of an XML document and parse it into a document. The document is validated if it declares a schema using a `validate-if-schema` option. Validation against a certain schema can be forced by passing a `$schema-hint` option. If the document is not well-formed, not valid, or fails to parse for another reason, the function throws an error.

Example:

```
(: parse the contents of the given element
  and return it as a document-node()
:)
xhive:parse(/channel/item[1]/content:encoded)

(: Take the given serialized document and store it in the DB :)
declare variable $doc-text as xs:string external;
```

```
let $doc := xhive:parse($doc-text, "http://www.w3.org/2005/Atom atom.xsd")
return xhive:insert-document("feed-lib/newentry.xml", $doc)
```

- **xhive:input() as document-node()**

This function returns the calling documents and is useful when there is another active context node.

- **xhive:java(\$class as xs:string, ...) as item()***

xDB offers an XQuery extension function interface that Java developers can use create new extension functions, which can then be called from XQuery using the **xhive:java** function.

Example:

```
xhive:java("com.mydomain.myclass", $x, doc("/mydoc")//item)
```

- **xhive:get-nodes-by-key(\$library as xs:string, \$indexname as xs:string, \$key as xs:string) as node()***

This function looks up the key in the index with the specified name on the specified library or document and returns the nodes in the index. This process provides direct access to indexes.

Example:

```
xhive:get-nodes-by-key("/MyLibrary", "item_index", "pc34")
```

- **xhive:full-path(\$document as node()) as xs:string**

This function returns the string that uniquely identifies the library child in the database. If the passed node is not a library child, the full path of the owner document is returned.

Example: If function `xhive:full-path($doc)` returns path `'/path/to/doc.xml'`, you can access the same document with function `doc('/path/to/doc.xml')`

- **xhive:document-name(\$document as node()) as xs:string**

This function returns the name of the document that is set by the `XhiveLibraryChildIf.setName(String name)` method. If the document has a name and when the passed node is a document or has an owner document, the name of the document is returned. Otherwise, this function returns an empty sequence.

- **xhive:document-id(\$document as node()) as xs:long**

This function returns the id of the document. This id is generated by xDB on creation of the document. If the passed node is a document or has an owner document, the id of the document is returned. Otherwise an empty sequence is returned.

- **xhive:force(\$items as item()* as item()***

This function forces the immediate evaluation of its argument. If this function is used as the outermost expression of a query, the query is evaluated immediately and the result is stored

internally. This function can make it possible to use the query result for modifying the searched data, which is normally impossible due to lazy query result evaluation.

Example:

```
xhive:force(doc('doc')//elem)
```

- **xhive:version(\$document as document-node()*, \$version as xs:string) as document-node()***

This function returns a document sequence that represent the contents of a set of specific input document versions. The function returns an empty sequence for nodes that are not documents, are not versioned, or for non-existing versions. The version argument is first evaluated as a label. If no version with that label is found, the argument is evaluated as a version ID. For example, if you have a version 1.4 which has a 1.2 label, a query for version 1.2 returns version 1.4.

It is possible to specify a set of documents as an argument. The following query retrieves all document version with the release2 label.

```
xhive:version(doc("/versioned-lib"), "release2")
```

- **xhive:version-property(\$document as document-node()*, \$version as xs:string, \$property as xs:string) as xs:string***

This function returns the value of a specified version attribute. This function is like the xhive:version function, but the result consists of a sequence of strings. The property argument must be one of the following:

- **date** or **creation-date** — The date on which the version was created, using the yyyy-mm-ddThh:mm format.
- **check-out-date** — The date on which the version was checked out, using the yyyy-mm-ddThh:mm format.
- **creator** — The name of the user who created the version.
- **checked-out-by** — The name of the user who checked out this document version.

- **xhive:version-date-property(\$document as document-node()*, \$version as xs:string, \$property as xs:string) as xs:date***

This function returns the value of a specified version date attribute. This function is similar to the xhive:version-property function, but the result consists of a sequence of xs:date values instead of xs:string values. In addition, the xs:date value is adjusted to timezone while the xs:string value is not adjusted to timezone. The property argument must be one of the following:

- **date** or **creation-date** — The date on which the version was created.
- **check-out-date** — The date on which the version was checked out.

- **xhive:version-ids(\$document as node()*[, \$branchversion as xs:string]) as xs:string***

This function returns the IDs of document versions. If no second argument is specified, all version IDs of the version space are returned as a string sequence. Passing a second argument, retrieves more detailed information:

- If the branch ID is specified, the result contains only those version IDs that are part of that branch and the ones shared with other branches.
- If **1** is passed as the argument, the result contains a list of all branch IDs in the version space of the document argument.
- If the version ID is specified, the result contains the version labels for that version.

For non-versioned documents, or when the branchversion argument refers to a nonexisting branch or version, the result is the empty sequence.

The example query below gets all the different titles of all book versions created before 2003:

```
distinct-values (
  let $doc := doc("/version-lib/book.xml")
  for $version in xhive:version-ids($doc)
  where xhive:version-property($doc, $version, "date") < "2003-01-01"
  return xhive:version($doc, $version)/book/title
)
```

- **xhive:version-id(\$document as node()) as xs:string**

This function returns the version ID of a specific document instance. Only applicable for documents marked as queryable. This is useful for retrieving versioning properties of documents returned by `xhive:collection-*-date()` functions.

Example:

```
for $doc in xhive:collection-after-date("/", xs:dateTime("2011-01-01T01:00:00Z"))
let $version-id := xhive:version-id($doc)
where $doc//name="John"
return <result>
  <name>{xhive:document-name($doc)}</name>
  <version-id>{$version-id}</version-id>
  <creator>{xhive:version-property($doc, $version-id, "creator")}</creator>
  <date>{xhive:version-date-property($doc, $version-id, "creation-date")}
</result>
```

- **xhive:collection-between-dates(\$path as xs:string, \$start as xs:dateTime, \$end as xs:dateTime) as node()***

xhive:collection-at-date(\$path as xs:string, \$date as xs:dateTime) as node()*

xhive:collection-before-date(\$path as xs:string, \$date as xs:dateTime) as node()*

xhive:collection-after-date(\$path as xs:string, \$date as xs:dateTime) as node()*

These functions return all document versions in the supplied path that match the supplied period, and are marked as queryable.

The example query below returns all document versions in some library that were created in the date range from Jan 1st 2011 up to May 13th 2012:

```
for $doc in xhive:collection-between-dates('/some/library',
  xs:dateTime('2011-01-01T01:00:00Z'), xs:dateTime('2012-05-13T10:00:00Z'))
```

```
return $doc)
```

Querying inside documents is also possible. The example query below returns all document versions in the date range that contained `<name>John</name>` in the XML:

```
for $doc in xhive:collection-between-dates('/some/library',
  xs:dateTime('2011-01-01T01:00:00Z'), xs:dateTime('2012-05-13T10:00:00Z'))
  where $doc//name='John' return $doc
```

Note: For regular documents or versioned documents that do not have the queryable option enabled, the function ignores the passed date parameters and always returns the current version, as the normal **fn:collection()** function would.

- **xhive:metadata(\$document as document-node)*, \$key as xs:string?) as xdt:untypedAtomic***

This function retrieves the value that belongs to the `$key` attribute in the metadata of the `$document` document. If the key is the empty sequence, the result is a sequence with the values of all metadata fields.

Example:

```
xhive:metadata(doc("/mydoc"), "author")
```

- **xhive:get-metadata-keys(\$document as document-node)* as xs:string***

This function retrieves all distinct keys of given documents(s).

Example:

```
xhive:get-metadata-keys(doc("/mydoc"))
```

- **xhive:highlight(\$arg as item)*, ...) as item)***

This function can be used for text highlighting. The xDB XQuery extension function **xhive:highlight(\$arg as item)*, ...) as item)*** is set on the query using the `XhiveXQueryQueryIf.setHighlighter(highlighter)` API.

When this function is called, the XQuery engine calls the extension function with the following arguments:

- The first extension function argument is a sequence of strings consisting of the tokens used by any full-text search in the current FLWOR expression.
- The second argument is a sequence of the positions of the tokens of the first argument. The position information can be used to distinguish phrases from single terms.
- The third argument contains the phrase ID of the token. All tokens of a phrase have the same ID. The ID is generated by a counter, the first phrase has ID = 1, the next 2 etc. The ID can be used to identify all tokens of a certain phrase.
- The last arguments of the highlighter extension function contain the `$arg` arguments passed to the `xhive:highlight` function.

All tokens, positions and phrase IDs have the same order, so the `nth` position of the positions argument and the `nth` phrase ID of the Phrase IDs argument belong to the `nth` token of the tokens argument.

For example, in the query

```
for $elem in //para
where $elem contains text "Rotterdam"
return xhive:highlight($elem)
```

the highlighter function is called with four arguments, token **Rotterdam**, position **1**, phrase ID **1** and the matching `para` element.

In the following phrase query

```
for $elem in //para
where $elem contains text {"Rotterdam", "harbour"} phrase
return xhive:highlight($elem)
```

the highlighter function is called with four arguments, **Rotterdam**, **harbour**, positions **1, 2**, phrase ID's **1, 1** and the matching `para` element.

- **xhive:created-at(\$Suri as xs:string) as xs:dateTime**

Returns when the document, library, or blob at **\$Suri** was created. If **\$Suri** does not exist, an error is raised (err:FODC0002).

- **xhive:last-modified(\$Suri as xs:string) as xs:dateTime**

Returns when the document, library, or blob at **\$Suri** was modified for the last time. If **\$Suri** does not exist, an error is raised (err:FODC0002).

- **xhive:child-documents(\$Suri as xs:string) as document-node()***

Returns direct document children of the library indicated by **\$Suri**, i.e., only documents that are located directly underneath the library, not recursive descendants like the built-in **doc** function. If **\$Suri** does not point to a library, or the library has no children, the empty sequence is returned. If **\$Suri** does not exist, an error is raised (err:FODC0002).

Note: Because xDB indexes cover all descendant documents, queries using this function will not be able to use any indexes on the library. In most cases, it might be better and easier to organize your content within xDB so that you do not need this function.

- **xhive:child-uris(\$Suri as xs:string) as xs:string***

Returns the absolute URIs of direct children of the library indicated by **\$Suri**. If **\$Suri** does not point to a library, or the library has no children, the empty sequence is returned. If **\$Suri** does not exist, an error is raised (err:FODC0002). In contrast to **xhive:child-documents**, this will return the URIs of blob nodes and/or libraries.

- **xhive:glob-documents(\$Suri as xs:string) as document-node()***

Returns all documents that match given wildcard database path **\$Suri**. The components of the path can specify the following wildcard characters:

- * - matches zero or more characters

– ? - matches one character

Relative database paths are resolved against the absolute path of the current context item.

Examples:

```
xhive:glob-documents ("/*")
xhive:glob-documents ("doc*.xml")
xhive:glob-documents ("/lib?/*/*")
```

Note: The **xhive:glob-documents** function does not use any indexes.

Using XQuery extension function *xhive:force*

The xDB XQuery extension function **xhive:force(\$items as item()*) as item()*** forces the immediate evaluation of its argument. For information about the function, see its [description](#).

Example:

```
String query = "xhive:force(doc('doc')//elem)";
XhiveXQueryResultIf result = lc.executeXQuery(query);
while (result.hasNext()) {
    XhiveXQueryValueIf value = result.next();
    // We know this query will only return nodes.
    Node node = value.asNode();
    // Remove this node
    node.getParentNode().removeChild(node);
}
```

Using XQuery extension function *xhive:highlight*

The xDB XQuery extension function **xhive:highlight(\$arg as item()*, ...) as item()*** is set on the query using the `XhiveXQueryQueryIf.setHighlighter(highlighter)` API. For information about the function, see its [description](#).

The analyzer that is used for the query processes the passed tokens. If the query strings contain wildcards, the analyzer replaces them with the characters specified in the `XhiveFtsUtilIf` interface. The `XhiveFtsUtilIf.compilePattern` method can be used to match query strings with wildcards against terms in the text.

Using indexes in XQuery

xDB has several index types that can be used to speed up XQueries:

- [Path indexes, page 151](#), [Multipath Indexes, page 153](#), [Value indexes, page 161](#), and [Element name indexes, page 167](#)

These index types are used when the optimizer determines that they apply to a specific expression in the query. Path and multipath indexes are most flexible and useful for typical queries.

- [Full text indexes, page 163](#)

Full text indexes can be used through the `xhiv:fts` function, see [Full-text queries, page 194](#).

- [Library name indexes and library ID indexes, page 165](#).

If available, these are always used by the `doc()` XQuery function.

- [ID attribute indexes, page 166](#)

The `id()` XQuery function always uses document ID attribute indexes. ID attribute indexes on libraries are never used by xqueries, except when explicitly used with the `xhiv:get-nodes-by-key()` extension function, [page 182](#).

Value and name element indexes

Value indexes and element name indexes are used for path expressions that:

- Start with a call to the XQuery `doc()` function that specifies one of the following:
 - The document or library containing the index.
 - One of the document or ancestor libraries.
 - A path expression that starts at a variable that has been bound to one or more libraries.

If the specified library does not contain a useful index, but one or more descendant libraries do, the evaluator uses the index on those descendant libraries. The query in the other libraries are evaluated by force search.

Example:

```
(: can use any indexes on root library or below :)
doc('//')//foo[@bar = 12]
```

```
(: can use indexes on "mylib" or below :)
doc('/mylib')//foo[@bar = 12]
```

```
(: can use indexes on "lib1" and "lib2" or below :)
(doc('/lib1'), doc('/lib2'))//foo[@bar = 12]
```

```
(: ditto, also works with declared and external variables :)
let $libs := (doc('/lib1'), doc('/lib2'))
return $libs//foo[@bar = 12]
```

```
(:
: will not use indexes on any libraries as the doc calls
: are expanded to the single documents below before the
: path expression is evaluated
:)
for $doc in (doc('/lib1'), doc('/lib2'))
return $doc//foo[@bar = 12]
```

- Contain descending steps, such as `child`, `descendant` or `descendant-or-self`, including the abbreviated versions like `//`. For element name indexes, the first step must be a `descendant(-or-self)` step.
- Contain at most one predicate per step. If there are multiple predicates in a step, they are changed to use `and`. For example, replace `//parent[@color = "red"][elem[@attr = "green"]]` with

`//parent[@color = "red" and elem[@attr = "green"]]` to have the query use possible value indexes on `parent/@color` and `elem/@attr`.

- Contain a predicate or where-clause that checks the indexed value. A value or general comparison is used against any expression that is constant for this path expression, and whose type corresponds to the type of the value index.
- Contain a step with an indexed element name.

Examples

The following examples use a value index with the default type "STRING" on the `attr` attribute of the `elem` element on the root library.

```
(: Use index without further checks :)
doc("/")//elem[@attr = $var]

(: Ditto :)
for $x in doc("/")//elem
where $x/@attr eq func(2)
return ...

(: Use index and check parent of indexed element :)
doc("/")//parent[@color = "red"]/elem[@attr = "green"]

(: Use index and check ancestors of indexed element :)
doc("/")/parent[@color = "red"]//elem[@attr = "green"]

(: Use index and lookup children :)
doc("/")//elem[@attr eq substring($str, 1, 3)]/name

(: Use index and return parent of indexed node :)
doc("/")//parent[elem/@attr = "black"]

(: Use index and return all ancestors of
   indexed nodes called "parent" :)
doc("/")//parent[descendant::elem/@attr = "black"]
```

With an element value index, the predicate or where-clause must check the contents of the element for an index. The following example uses an element value index on the `elem` element in the root library.

```
(: Use the index directly :)
doc("/")//elem[. eq "red"]

(: In a flower expression :)
for $x in doc("/")//elem
where $x = "green"

(: Uses text() instead of context node :)
doc("/")//person/elem[text() = "yellow"]

(: Use the index and return the parent of the indexed node :)
doc("/")//person[elem = "black"]

(: Or equivalently :)
for $x in doc("/")//person
where $x/elem = "black"
return $x
```

Range queries

Range queries are queries that constrain data to a range of values, instead of to a single value. If the optimizer finds a predicate or where-clause that uses both **less or equal** and **greater or equal** on the same node, it can use an index to find the values in the requested range. For example:

```
doc('//')//book[@author >= "A" and @author < "B"]
```

If there is an index with sorted keys on `book/@author` element, the optimizer scans the index from A to B to find the result of this expression.

The conditions refers to the same node, for example:

```
(: Cannot use range query on author index :)
doc('//')//book[author >= "A" and author < "B"]
```

The optimizer cannot use the `author` index in a range query. The book could have one author satisfying the first condition and another author that satisfies the second condition. To make both conditions refer to the same author and allow use of the index,

```
(: Can use range query on author index :)
doc('//')//book[author[. >= "A" and . < "B"]]

(: Can use path index book[author + title] :)
doc('//')//book[author = "Asimov" and title[. >=
"Robot" and . < "Second"]]
```

Indexing metadata

The following example uses the `doc()` function and an index that has been created on the `mylib` library and the `author` metadata field.

```
doc("/mylib")[xhive:metadata(., "author") = "Jane Doe"]
```

The function looks up `Jane Doe` in the `author` index and returns all matching documents. The optimizer can only use indexes for expressions where the metadata key is a string literal or the literal empty sequence, not a generic expression.

Full text indexes would use an expression like the following:

```
doc("/mylib")[xhive:fts(xhive:metadata(., "p"), "XQuery")]
```

Multiple indexes

Different parts of a query can use different indexes. The following example uses an index on attribute `x` of element `x`, and an index on attribute `y` of element `y`. If both indexes can be used for a single path expression, the optimizer creates a query plan with an intersection.

Examples

```
for $x in doc("/")//x[@x = "x"]
for $y in doc("/")//y[@y = $x/@yref]
return ...
```

```
(: or, equivalently :)
```

```
for $x in doc("/")//x
for $y in doc("/")//y
where $x/@x = "x"
and $y/@y = $x/@yref
return ...
```

In the following example, the optimizer first looks up "x" in the index for `x/@x` and stores the result in a temporary set. Then the optimizer looks up "y" in the index for `y/@y` and checks that the parents of the indexed elements are present in the temporary set. If the parents are present, the node element is added to the result set.

```
doc("/")//x[@x="x"]/y[@y="y"]
```

The following query returns similar results:

```
let $x := xhive:get-nodes-by-key("/", "x/@x", "x")
return xhive:get-nodes-by-key("/", "y/@y", "y") [parent::x intersect $x]
```

Metadata indexes can also be used to create similar combinations, as described in the following example.

```
doc("/") [xhive:metadata(., "status") = "ready" and ./x/@x = "x"]
```

```
doc("/") [xhive:metadata(., "author") = "PP"]
//chapter[xhive:fts(title, "xDB")]
```

Indexes and order by

Indexes can be used to speed up queries that use an **order by** statement, if:

- The expressions in the **order by** statement match the indexed values in the correct order.
- The FLWOR expression is run on a single indexed library or document.
- The **order by** statement is not stable.

Examples

If there is a multi-valued path value index, the optimizer tries to use as many values from the index as possible. In this case the order specs have to be in the same order as in the index specification.

```
(: with an index on foo.xml, this will use an order by index :)
for $book in doc('foo.xml')//book[@year]
  (: have to mention child for index usage :)
order by $book/@year descending
return $book
```

Enable [queryplan-debug](#) if you want to verify whether an order by query is being optimized. For example, a path value index like `//book[@year<STRING> + title<STRING>]` generates an output like `Found an index to support the first 2 order specs.`

```
declare option xhive:queryplan-debug 'true';
for $book in doc('foo.xml')//book[@year and title]
order by $book/@year, $book/title
```



```
return $book
```

If the query plan does not match the expected plan, the optimizer debug statements are enabled to check whether the optimizer considered the desired index.

```
declare option xhive:optimizer-debug 'true';
for $book in doc('foo.xml')//book[@year and title]
order by $book/@year, $book/title
return $book
```

It is also possible to optimize a subset of the order specs. For example, if an index can only support two of three order specs, only the last order spec is evaluated, and only in the case the first two values are equal.

Queries that use range or equality comparisons on index values in combination with an order by statement also benefit from indexes. With the path value index from the last example, the following query is faster.

```
for $book in doc('/booklib/')//book[@year = '2002' and title > 'V']
order by $book/@year, $book/title
return $book
```

Proprietary XQuery extension to order by

xDB implements a proprietary extension to **order by** statements that makes it possible to order the results of a query depending on user input.

The XQuery FLWOR grammar is extended as follows:

```
orderModifier:
  ("ascending" | "descending"
  | <"ascending" "if" "(" ExprSingle ")">)?
  (<"empty" "greatest"> | <"empty" "least">)? ("collation" URILiteral)?
```

If the result value is **true**, the expression in parenthesis is evaluated to a Boolean value and the result is ordered ascending. If the result value is **false**, the result is ordered descending.

This syntax can be useful for writing queries that require ordering data by many different columns, depending on user input. Imagine ordering tabular data with 8 columns in all descending/ascending combinations by writing 64 different queries and encapsulating them in if statements.

```
declare variable $asc_order1 external;
declare variable $asc_order2 external;

for $entry in //...
order by $entry/id ascending if ($asc_order1),
        $entry/name ascending if ($asc_order2)
return
  $entry
```

Together with dynamic checks for QNames (e.g. `$entry/*[node-name() eq $order_coll]` instead of `$entry/id`) you can avoid writing duplicated query code. This functionality is proprietary and queries using it are not compatible with other XQuery implementations. The ascending if .. syntax prevents index supported evaluation of order by expressions.

Using type information in XQuery

xDB can store type information in two ways:

- Value indexes can have a [type](#), [page 161](#).
- During validation of a document, PSVI information can be stored with the nodes of the document. The type of the node is persisted as declared in the associated XML schema.

This value index type information is used in XQuery. For example, in the query

```
//element[@id < /my/first/idelement]
```

the way the comparison between the `id` attribute and the `idelement` attribute is processed, depends on the type-information as follows:

- If no type information is found, the comparison is performed between the string values of the two attributes.
- The PSVI information is stored for the attributes. For example, if the `/my/first/idelement` item is stored as an integer, the comparison is performed as if both attributes are integers.
- If the `id` attribute is stored as an integer, but `/my/first/idelement` is stored as an incompatible type, the XQuery processor throws an exception.

The index type never determines the type used in the comparison. The type for the comparison is determined first, then the matching index type is used. However, when a typed index is used it can lead to different query results compared to a query evaluation without that index. The `id` attributes are treated as integers when an integer index is used, but they are treated as text in case of an untyped index.

It can be useful to enable debugging for a query by setting the [xhive:queryplan-debug option](#) to **true**. The debugging information includes which indexes are used.

It is possible to execute the sample XQuery with an integer comparison by explicitly casting the compared value to the right type, or using one of the internal conversion functions. The sample XQuery can be executed even if the `/my/first/idelement` does not have PSVI information or has PSVI information but is of a non-integer type in the linked XML Schema, as follows:

```
//element[@id < xs:int(/my/first/idelement)]
```

or

```
//element[@id < (/my/first/idelement cast as xs:integer)]
```

XQuery full-text search

xDB partially implements the W3C XQuery Full-Text Facility Standard, with some extensions. For information on limitations, see [Full-Text search limitations](#), [page 195](#).

This section provides descriptions and usage examples of:

- [full-text logical operators](#), [page 195](#)
- [wildcard options](#), [page 195](#)
- [fuzzy option](#), [page 196](#)
- [thesaurus option](#), [page 196](#)
- [anyall options](#), [page 197](#)

- [positional filters, page 197](#)
- [cardinality option, page 198](#)
- [score variables, page 198](#)

xDB supports the full-text operator `contains text`. Earlier versions of the XQuery Full-Text specification used the keyword `ftcontains`. This is still supported by xDB, but deprecated.

xDB also extends XQuery with a [proprietary full-text search function, page 198](#). This is still supported in xDB 10, but the standard syntax proposed by W3C is preferred.

Full-text search limitations

The current full text search implementation has certain limitations:

- The query parser recognizes the boost factor, but it is not possible to rank the results.
- Prefix queries are not passed through the analyzer. If an index contains only lowercase terms, uppercase letters are not used in a prefix query.
- Using phrase queries on indexed nodes when the index does not support phrase queries generates an unsupported operation exception.
- Phrase queries are always be surrounded by double quotes. The query parser does not recognize a list of terms within single quotes as a phrase query.

Full-text logic operators

xDB supports the `ftor`, `ftand`, `ftnot`, and `not in` full-text logic operators. For descriptions of these operations see http://www.w3.org/TR/xpath-full-text-10/#logical_ftoperators.

Examples of full-text search queries with logic operators

```
(: retrieves all books with title containing terms "programming" and "web" :)
doc('bib.xml')/bib/book[title contains text "programming" ftand "web"]
```

```
(: retrieves all books with title containing terms "Unix" and "TCP"
   or term "programming" :)
doc('bib.xml')/bib/book[title contains text "Unix" ftand "TCP" ftor "programming"]
```

```
(: retrieves all books with title containing terms "Unix",
   but not containing term "UDP" :)
doc('bib.xml')/bib/book[title contains text "Unix" ftand ftnot "UDP"]
```

```
(: retrieves all books with title containing terms "Unix"
   when it is not part of "Unix environment" :)
doc('bib.xml')/bib/book[title contains text "Unix" not in "Unix environment"]
```

Queries with wildcards

xDB supports the `.`, `.*`, `.+`, and `{n,m}` wildcard qualifiers. For descriptions of full-text wildcard options, see <http://www.w3.org/TR/xpath-full-text-10/#ftwildcardoption>.

Examples of full-text search queries with wildcard options

```
(: retrieves all books with publisher containing term starting from "Kauf" :)
doc('bib.xml')/bib/book[publisher contains text "Kauf"]
```

```
doc('bib.xml')/bib/book[publisher contains text "Kauf.*" using wildcards]

(: retrieves all books with publisher containing term starting
   from "Aca" then followed by arbitrary character and ended by "emic" :)
doc('bib.xml')/bib/book[publisher contains text "Aca.emic" using wildcards]

(: retrieves all books with publisher containing term "Publisher"
   or terms starting from "Publisher" and ended by arbitrary character :)
doc('bib.xml')/bib/book[publisher contains text "Publisher.?" using wildcards]

(: retrieves all books with publisher containing term "Aca.emic" :)
doc('bib.xml')/bib/book[publisher contains text "Aca.emic" using no wildcards]
```

Queries with fuzzy search

xDB extends XQFT functionality with the `xhive:fuzzy` search option, which is supported as an extension option of XQFT standard. When using the `xhive:fuzzy` option, you should specify a minimum similarity number in the range [0, 1] which defines the similarity between query term and searching term. The similarity is defined according to Levenshtein distance formula.

Example of full-text search queries with fuzzy option

```
(: retrieves all books with publisher containing term "Daufman" or similar to it.
   The minimum similarity value is 0.8. :)
doc('bib.xml')/bib/book[publisher contains text "Daufman" using
   option xhive:fuzzy "similarity=0.8"]
```

Queries with thesaurus

xDB supports full text xqueries with thesaurus option. For a description of the XQFT thesaurus option, see <http://www.w3.org/TR/2011/REC-xpath-full-text-10-20110317/#ftthesaurusoption>.

To support thesaurus in xqueries, a thesaurus handler must be specified. This can be done in your application, or in your XQuery using `xquery` option `xhive:fts-thesaurus-class`. If a thesaurus handler is set by both, the option takes precedence.

Example of a full-text search query with thesaurus

```
(: retrieves all p elements which contain the term 'duty'. :)
for $p in doc('/data/fti-books-document.xml')//p
  where $p contains text 'duty'
  using thesaurus at 'http://www.emc.com/usability' relationship 'UF'
return $p
```

Queries with thesaurus handler

As neither schema nor storage location of thesauri are fixed, a class implementing interface `com.xhive.query.interfaces.XhiveThesaurusHandler` is required to support thesaurus in xqueries.

There are two ways to set the thesaurus handler:

- method `setThesaurusHandler(XhiveThesaurusHandlerIf thesaurusHandler)` of interface `com.xhive.query.interfaces.XhiveXQueryQueryIf`
- xquery option `xhive:fts-thesaurus-class`

The advantage of using the API to set the thesaurus handler is the ability to initialize the class with additional context information.

If a thesaurus handler is set by both API and xquery option, the option takes precedence.

Samples

[XQueryWithThesaurus.java](#)

[MyThesaurusHandler.java](#)

API documentation

[com.xhive.query.interfaces.XhiveThesaurusHandlerIf](#)

Anyall options

xDB supports the **any**, **any word**, **all**, **all words**, and **phrase** anyall options. For descriptions of anyall options, see <http://www.w3.org/TR/xpath-full-text-10/#ftwords>.

Examples of full-text search queries with anyall options

```
(: retrieves all books with title containing phrase "TCP programming"
and term "UDP" :)
doc('bib.xml')/bib/book[title contains text {"TCP programming", "UDP"} all]

(: retrieves all books with title containing phrase "TCP programming" :)
doc('bib.xml')/bib/book[title contains text {"TCP", "programming"} phrase]

(: retrieves all books with title containing at least one of "TCP",
"programming" or "UDP" term :)
doc('bib.xml')/bib/book[title contains text {"TCP programming", "UDP"} any word]
```

Positional filters

xDB supports the **ordered**, **window distance**, and **anchoring** positional filters. For a description of positional filters, see <http://www.w3.org/TR/xpath-full-text-10/#ftposfilter>.

Examples of full-text search queries with positional filters

```
(: retrieves all books with title containing both "unix" and
"programming" and the order of matched terms is the same as in the query :)
doc('bib.xml')/bib/book[title contains text "unix" fband "programming" ordered]

(: retrieves all books with title containing both "unix" and
"programming" which are found within 3 words unit :)
doc('bib.xml')/bib/book[title contains text "unix" fband "programming" window 3 words]
```

```
(: retrieves all books with title containing both "unix" and "programming" and
the distance between matched terms must be at least 2 words :)
doc('bib.xml')/bib/book[title contains text "unix" ftand "programming"
                        distance at least 2 words]
```

```
(: retrieves all books with title containing both "unix" and
"programming" and the matched tokens cover start and end positions of the title :)
doc('bib.xml')/bib/book[title contains text "unix" ftand "programming" at start at end]
```

Cardinality option

xDB supports the **cardinality** option. For a description of cardinality option, see <http://www.w3.org/TR/2010/CR-xpath-full-text-10-20100128/#fttimes>

Examples of full-text search queries with cardinality option

```
(: retrieves all books which have authors containing
at least 2 tokens "Serge" in the name :)
doc('bib.xml')/bib/book[author contains text "Serge" occurs at least 2 times]
```

```
(: retrieves all books wich have title containing
exactly 1 token "Web" in the name :)
doc('bib.xml')/bib/book[title contains text "Mike" occurs exactly 1 times]
```

Score variables

xDB supports a scoring mechanism using score variables in **for** and **let** clauses of FLWOR expressions. Score variables are xs:double types in the [0, 1] range. A higher score value implies a higher degree of significance. For a description of XQFT score variables, see <http://www.w3.org/TR/xpath-full-text-10/#section-score-variables>.

Examples or full-text search queries with score variables

```
(: retrieves all books with title containing both "unix" and
"programming" terms and sorted by score :)
for $book score $s in doc('bib.xml')/bib/book[
title contains text "unix" ftand "programming"]
order by $s
return $book
```

```
(: retrieves all books with title containing "unix" and sorted by
score in descending order.
However, the scores reflect whether the book's content contains
"programming" and "java" terms :)
for $book in doc('bib.xml')/bib/book[title contains text "unix"]
let score $s := $book/content contains text "programming" ftand "java"
order by $s descending
return $book
```

Score calculation

Scoring is available for both indexed and non-indexed queries. When using indexes, the quality of the score estimation is much higher. Depending on the options that were used, xDB has access to frequency and occurrence counts for the node set that was searched.

For optimal score estimation, full-text indexes are created with the `FTI_SUPPORT_PHRASES` and `FTI_SUPPORT_SCORING` [full-text index options, page 163](#).

The scoring implementation of xDB is partially based on Lucene. xDB also uses a Lucene-based similarity class, that lets the user influence the results by changing the similarity measures using the XQuery option `xhive:fts-similarity-class`.

For more information about the concepts used to estimate a query score, see the Lucene [Similarity API](#).

The most significant difference between the xDB scoring implementation and the Lucene implementation is that xDB does not evaluate all results. xDB estimates all scores before returning the first result and first score. xDB estimates the expected number of results and uses the amount to normalize and weight different query components.

Note: In the current xDB implementation it is not possible to increase the weight of a node manually, and thus to increase the relevance of that node with respect to scoring. xDB cannot guarantee the scoring relevance order stability.

Boost scoring models

xDB supports boosting scoring models to boost results of the full-text query coming from the library with higher priority (e.g. freshness criteria). So, if a user performs some full-text search and gets results ordered by score then the user can set a special callback which would define the boosting parameters for the specific library and XQuery engine will recalculate the final score for the resulting documents applying boosting parameters and returning results ordered by the modified scores.

xDB supports the following boosting models:

- Weighted boosting model
- Factor/shift boosting model

Weighted boosting model

The weighted model defines a new score according to the formula:

$$finalScore = ((origScore * (100 - freshnessWeight) + libFreshness * freshnessWeight))/100;$$

- *origScore* is the original score coming from xDB engine;
- *freshnessWeight* is the weight of the library freshness boost (should fit into [0,100] range);
- *libFreshness* is the freshness of the library (should fit into [0, 1] range).

Factor/shift boosting model

The factor/shift model defines a new score according to the formula:

$$finalScore = * origScore * factor + shift;$$

- *origScore* is the original score coming from xDB engine;
- *factor* defines the factor value for the library;
- *shift* defines the shift value for the library.

Using boost scoring models

The *weighted boosting model* and the *factor/shift boosting model* can be used with xDB as boosting scoring models for full-text XQuery results.

For weighted scoring, use *XhiveWeightedFreshnessBoostIf* to set the scoring callback for the query. The code fragment below shows how to define and set the callback.

```
...
XhiveXQueryQueryIf query = Library.createXQuery(QUERY);
XhiveWeightedFreshnessBoostIf weightedScoresCallback =
    new XhiveWeightedFreshnessBoostIf() {

    @Override
    public WeightedFreshnessParameters
        getBoostParameters(final XhiveLibraryChildIf libraryChild) {

        return new WeightedFreshnessParameters() {

            @Override
            public double getFreshnessWeight() {
                return 40;
            }

            @Override
            public double getLibraryChildFreshness() {
                return someFreshnessFunction(libraryChild);
            }
        };
    };

query.setLibraryWeightedFreshnessBoost(weightedScoresCallback);
...
```

Use *XhiveScoreBoostFactorIf* to set the factor/shift scoring callback for the query. The code to define and set the shift/factor callback is analogous to the weighted model callback.

Samples

[BoostLibraryScore.java](#)

API documentation

[com.xhive.query.interfaces.XhiveWeightedFreshnessBoostf](#)

[com.xhive.core.interfaces.XhiveScoreBoostFactorlf](#)

Using the xhive:fts full-text search function

xDB extends XQuery with a proprietary full-text search function that can be used to search for *'terms'* within a text string. It can use indexes, allows wildcards and prefixes and allows searching for exact or sloppy phrases. The xDB full-text search implementation is partially based on code from the [Lucene](#) project.

In xDB, the basic units for full-text indexing and searching are called *'terms'*. In general, you can think of terms as words. For example, the string *'yadda yadda yadda'* contains three terms, each with the value *'yadda'*. The [analyzer](#) determines what is regarded as a term.

The xDB full-text search function is declared as follows:

```
xhive:fts (node(s), querystring, options)
```

The first argument of the function is a node or a set of nodes. If a set of nodes is provided, the full-text search is executed on all the nodes. The second argument is a query string. The options argument is optional, and is a string literal containing a list of options separated by semicolon. The available options are:

- `include-attrs` - if set, also search on the attribute values of searched elements (and descendants), along with other text nodes. **Note:** Use of the `include-attrs` option in combination with full-text `index(es)` requires using the `FTI_INCLUDE_ATTRIBUTES` option on the `index(es)`.
- `analyze-wildcards` - if set, the terms in the query will be sent to the analyzer. **Note:** If the query contains wildcards, the analyzer must not remove the characters used to represent wildcards.
- `analyzer classname` - if set, specifies the analyzer to be used on the query text.

The result is a boolean value, returning true if the text matches the query.

xDB full-text search query syntax

The syntax of xDB full-text search queries is as follows:

```
Query      ::= Clause ( [ Conjunction ] Clause ) *
Conjunction ::= 'AND' | 'OR' | '||' | '&&'
Clause     ::= [ Modifier ] BasicClause [ Boost ]
Modifier   ::= '-' | '+' | '!' | 'NOT'
BasicClause ::= ( TermQuery | Phrase | '(' Query ')' )
```

```
TermQuery      ::= ( Term | WildCardTerm | PrefixQuery ) [ Fuzzy ]
PrefixQuery    ::= Term '*'
Phrase         ::= '"' Term * '"' [ SlopFactor ]
Fuzzy          ::= '~'
SlopFactor     ::= '~' DecimalDigit+
Boost          ::= '^' DecimalDigit+ '.' DecimalDigit+
Term           ::= <a-word-or-token-to-match>
WildCardTerm  ::= <a-word-or-token-to-match-with-wildcards>
```

The following characters are reserved: + - ! () : ^ [] " { } ~ * ?

If used without special meaning, they must be escaped with a backslash (\) .

The syntax provides for various kinds of searching, including:

- [Boolean queries](#)
- [Prefix searches](#)
- [Phrase searches](#)
- [Searches with wildcards](#)

Boolean queries

A Boolean query represents a composite query that can contain subqueries of arbitrary nesting level, with composition rules such as **and**, **or**, **not**.

For each subquery of a boolean query, two binary qualifiers control how its superquery is matched:

- **prohibited** - if set, the superquery is a match only when the subquery does not match. A subquery can be marked as prohibited using modifier -, !, or **NOT**.
- **required** - if set, the superquery is a match only when the subquery does match. This condition is necessary but not sufficient for the superquery to match. Queries can be marked as required using modifier +.

The default implicit conjunction is **OR**. For example, by default, the query "apples oranges bananas" is equal to "apples OR oranges OR bananas". The implicit conjunction can be changed locally using the `xhive:fts-implicit-conjunction` option. For example, the query

```
//element[xhive:fts(., "apples AND oranges")]
```

generates the same results as the query

```
(# xhive:fts-implicit-conjunction 'AND' #) {
//element[xhive:fts(., "apples oranges")]
}
```

There is some overlap of functionality with XQuery. For example, the query below also generates the same results:

```
//element[xhive:fts(., "apples") and xhive:fts(., "oranges")]
```

Although there is no semantic difference between the three, in the current implementation, queries like the first two can be significantly faster than queries like the last one.

Prefix searches

A prefix search searches for all terms starting with a certain prefix.

Phrase searches

A phrase query represents a query that is matched against a consecutive sequence of terms in the field. A phrase query can have an optional *boost* factor and an optional *slop* parameter. The *slop* parameter can be used to relax the phrase matching by accepting out of order term sequences. For example, the phrase query 'winding road' matches 'winding road' but not 'road winding', unless used with more relaxed *slop* factors.

Searches with wildcards

xDB allows using the * and ? characters as wildcards in searches. The * wildcard is a substitute for an arbitrary number of characters, the ? wildcard substitutes a single character.

Only indexes built with the option `FTI_LEADING_WILDCARD_SEARCH` are suitable to search for terms with a wildcard as the first character. If this option is not set, the search can become extremely slow.

The Analyzer

The process of breaking content in terms (words) is called *tokenization*. In xDB, tokenization is done by an *analyzer*. An analyzer breaks up content in tokens, and can also change tokens to improve the search capacity. For example, an analyzer can change terms to lowercase, or change a term from plural to singular.

Both the searched text and the query are passed through the same analyzer. If an index is available, the same analyzer used for building the index is used for analyzing the query. If no index is available, the value of the `fts-analyzer-class` option determines which analyzer is used. To use a different analyzer in the query, the analyzer class name must be included in the options argument of the `xhive:fts` function.

The default analyzer for the `fts` function:

- Analyzes the input string as a list of *terms*, not as a list of characters. (For example, the `contains` function in XQuery considers the text as a single monolithic string.)
- Creates terms containing only letters and/or digits. Everything else triggers the start of a new term.
- Converts all characters in a term to lower case.

- Filters out the English stopwords "a", "and", "are", "as", "at", "be", "but", "by", "for", "if", "in", "into", "is", "it", "no", "not", "of", "on", "or", "s", "such", "t", "that", "the", "their", "then", "there", "these", "they", "this", "to", "was", "will", "with".

XQuery performance tuning

The following actions can improve XQuery performance:

- If a path expression uses an index, use as few explicit steps as possible because all steps must be checked to verify whether they match. For example: (: Preferred with index :)

```
doc("/")//elem[@attr = "green"]
```
- If a path expression does not use an index, but must be searched by scanning the document, use as many explicit steps and predicates as possible. That way, branches of the DOM tree can be skipped as soon as possible. For example: (: Preferred without index :)

```
doc("/")/docelem/persons/person/name/elem[@attr = "green"]
```
- Some predicate expressions are optimized to stop searching as soon as the required number of items are found. If the predicate is an integer expression that does not depend on the context node, context position or context size evaluation stops as soon as possible. The same applies if the predicate requires the `position()` function to be less than or equal to such an expression. For example:

```
(: Stops searching after the second foo element :)
let $x := doc("/mydoc")//foo
return $x[position() le 2]
```

```
(: Also stops searching after the second foo element :)
let $x := doc("/mydoc")//foo
return $x[2]
```

Sometimes a query can be modified to use such a predicate. For example:

```
(: Tests predicate for all foo elements :)
(doc("/mydoc")/descendant::foo)[position() = 2 to 4]
```

```
(: Stops after the fourth foo element :)
(doc("/mydoc")/descendant::foo)[position() le 4][position() ge 2]
```

It is also possible to use the `subsequence()` function to make sure that the search does not continue after the requested number of items.

```
(: Stops searching after the fourth foo element :)
let $x := doc("/mydoc")//foo
return subsequence($x, 2, 3)
```

- Use the `let` command to move expensive computations out of loops. For example:

```
(: Before (searches document b for each occurrence of element a) :)
doc("/a")//a[@id = doc("/b")//b[@id = "10"]/@ref_a_id]
```

```
(: After (searches document b at most once) :)
let $ref_a_id := doc("/b")//b[@id = "10"]/@ref_a_id
return doc("/a")//a[@id = $ref_a_id]
```

- If something occurs only once, a predicate [1] allows the evaluator to stop searching after the first occurrence. For example:

```
(: Even better if you know each id is only used once,
   will stop searching after 1st occurrence found. :)
let $ref_a_id := (doc("/b")//b[@id = "10"])[1]/@ref_a_id
return (doc("/a")//a[@id eq $ref_a_id])[1]
```

```
(: Do not confuse the previous query with this one.
   This query is probably not what the user intended. :)
let $ref_a_id := doc("/b")//b[@id = "10"][1]/@ref_a_id
return doc("/a")//a[@id eq $ref_a_id][1]
```

- Use the `unordered` function where possible. If the order of the result is not important, using `unordered` can speed up the query by allowing the evaluator to skip sorting of the result in document order. For example:

```
(: Assuming the query uses an index, this needs to
   sort the values looked up in the index. :)
doc("/")//elem[@attr = 'value']
```

```
(: This version needs no sorting step. :)
unordered(doc("/")//elem[@attr = 'value'])
```

- Recursive functions are tail recursive. xDB implements the *tail call modulo cons* recursion generalization. Tail recursion saves stack space by allowing recursive functions calls that return the result of the call directly to be evaluated iteratively, without any recursion. Tail recursion does only work if the tail call is the last item in an evaluation branch of the function, except for other tail calls. Tail recursion also works for mutually tail-callable functions. For example:

```
(: This function is tail recursive because the recursive call is
: the last thing on the 'else' evaluation branch
:)
declare function local:x($arg)
{
  if ($arg eq 3) then 'foo'
  else ('a', local:x($arg - 1))
};
(:
: This function is not tail recursive because the result of the
: recursive call is used in the 'or' statement and thereby needed
: for evaluation of the method body.
:)
declare function local:x($arg)
{
  exists($arg/@attr) or local:x($arg/child::* )
};
```

Many functions can be modified to be tail recursive. For example:

```
(: Not tail recursive because the result is used in the '+' operation :)
declare function local:sum($x as xs:integer) as xs:integer
{
  if ($x eq ) then 0
  else $x + local:sum($x - 1)
};
```

```
(: Re-written to use an accumulator, call with $acc = 0 to start :)
declare function local:sum($x as xs:integer, $acc as xs:integer)
  as xs:integer
{
  if ($x eq ) then $acc
  else local:sum($x - 1, $x + $acc)
};
```

Tail recursion does not necessarily improve performance, but it allows recursive functions that would otherwise result in stack overflows. Because of the evaluation strategy, the results of tail calls cannot be type checked and the parameters to tail calls are not evaluated lazily but directly. The result of the method is still type checked.

To verify if a function is tail recursive, enable the [queryplan-debug option](#): declare option `xhive:queryplan-debug 'stdout'`; . The maximum tail recursion depth can be set using the `xhive:max-tail-recursion-depth` option, as described in [XQuery options, page 179](#).

XQuery collation support

In xDB, a collation consists of a locale and an optional strength, separated by a slash. xDB's collation support relies on Java's built in support for locales and uses collators from IBM's ICU package (included in the distribution). The class `java.util.Locale` specifies which locales are supported. A list can be created using a code fragment as in the following example.

```
java.util.Locale[] locales = java.util.Locale.getAvailableLocales();
for (int i = 0; i < locales.length; ++i) {
  System.out.println(locales[i].toString());
}
```

The strength argument is a number from 1 to 4, corresponding to the collator strengths `PRIMARY`, `SECONDARY`, `TERTIARY`, and `IDENTICAL`. If the strength is unspecified, the default strength of the `java.text.Collator` instance is used, as shown in the following example.

```
compare($string1, $string2, xs:anyURI("en")),
starts-with($string, "abc", xs:anyURI("nl_NL")),
ends-with($string, "xyz", xs:anyURI("no/1")),
substring-after($string1, $string2, xs:anyURI("fr_CA/2"))
```

If no collation is specified, the implementation uses the normal Java String class methods for comparison, effectively comparing UTF-16 code units. Functions such as `substring` and `string-length` always count Unicode code points, which does not necessarily produce the same result as counting Java characters.

XQuery Profiler

To help you find out why a query runs slow, how long parts of an XQuery take to execute, or how much data is read, you can create a profile of an XQuery execution.

The Admin Client provides a simple graphical user interface for [profiling XQueries, page 243](#).

Profiling produces an XML document containing the original query text and a tree of XML nodes that represent the functions, modules, variables and expressions of the XQuery.

When an XQuery is executed with the `xhive:profile` option enabled (either as an XQuery option, or by your application), after at least partial execution the XQuery will contain profiling information.

XQuery profiling methods

xDB provides a static XML representation of an XQuery through the query plan API methods:

- **XhivePreparedQueryIf.getQueryPlan(DOMImplementation)**
- **XhiveXQueryQueryIf.getQueryPlan()**
- **XhiveXQueryResultIf.getQueryPlan()**

These methods return an XML document containing the original query text and a tree of XML nodes that represent the functions, modules, variables and expressions of the XQuery.

With the `xhive:profile` option enabled (as an XQuery option, or through the XQuery compiler API), after at least partial execution, calling **XhiveXQueryResultIf.next()** will return profiling information.

The `pagesRead` attribute will only be accurate if the query is run in a "new" session immediately after a call to `begin()`.

XQuery implementation

The xDB XQuery implementation is based on the [XQuery 3.0 W3C Candidate Recommendation \(08 January 2013\)](#) specification. xDB implements the **Full Axis Feature** and provides the ancestor, ancestor-or-self, following, following-sibling, preceding, and preceding-sibling axes.

XQuery 3.0 optional features

Supported optional features:

- Full Axis Feature
- Schema Aware Feature
- Module Feature
- Higher-Order Function Feature

Unsupported optional features:

- Static Typing Feature
- Serialization Feature

Unsupported XQuery 3.0 features and functions

Unsupported features:

- Forwards references to global variables
- Allow modules to reference each other without restriction
- Union of lists
- Namespace nodes

Unsupported functions:

- fn:format-date
- fn:format-time
- fn:format-dateTime
- fn:format-integer
- Serialization option 'html' in combination with fn:serialize
- Context dependent functions fn:position(), fn:size() and fn:last() outside of a predicate

As in XPath 1.0, a path step does not set the context size and position, only predicates do so. A work-around is to use a **for** iterator with a positional variable and the **count()** function.

The following xhive functions cannot be used as named function references:

- xhive:fts
- xhive:java
- xhive:metadata
- All functions of the proprietary XQuery Update Syntax

XQuery full-text query support

xDB partially implements the W3C XQuery Full-Text Facility Standard available at <http://www.w3.org/TR/xpath-full-text-10/>. For more information, refer to the section on [XQuery full text search](#), page 194.

Collation support

Several XQuery functions take a collation argument. The possible values of this argument are implementation defined, according to the XQuery specification. In xDB, a collation consists of a locale

and an optional strength, separated by a slash. xDB's collation support relies on Java's built in support for locales and uses collators from IBM's ICU package (included in the xDB distribution).

XQuery Security Features

xDB's XQuery implementation has several features that are security sensitive, including:

- access to files in arbitrary locations through the `doc()` and `collection()` functions
- updating XML content (only in read/write transactions)
- execution of imported XQuery code through module imports
- importing XML schema files
- execution of arbitrary Java code and Java module imports

Applications can control these by implementing a security policy.

XQuery security methods

Applications can control which xQuery security features are enabled in an XQuery by implementing the Java interface `XhiveXQueryPolicyIf` (or better, subclassing `DefaultXhiveXQueryPolicy`) and registering it using `XhiveXQueryCompilerIf.setSecurityPolicy(XhiveXQueryPolicyIf)`. This allows fine-grained control over each feature, including which particular Java classes or URLs may be accessed.

Like SQL, XQuery is vulnerable to injection attacks when user input is inserted by string concatenation. To avoid XQuery injection attacks, always use [external variables, page 175](#).

Samples

[XQueryCompiler.java](#)

API documentation

[com.xhive.query.interfaces.XhiveXQueryPolicyIf](#)

[com.xhive.query.interfaces.DefaultXhiveXQueryPolicy](#)

XQuery modules

xDB implements the **Module Import Feature** for creating library functions. Modules can be imported using the following syntax:

```
import module namespace prefix = 'http://some/namespace/uri' at 'location';
(: ... use functions from the module ... :)
```

XQuery modules have the following characteristics:

- The XQuery module location depends on the implementation definition. In xDB, the location part can be any valid Java URI, for example `file://...` or `http://...`, as well as a URI within the database. Using `xhive://` or a relative or absolute path without a protocol identifier follows the same syntax

as the `doc()` function. Import paths are evaluated relative to the `XhiveNodeIf` interface or the library in which the query is executed or created.

- Importing a module into a current query makes available all functions and variables that have been declared within the module namespace.
- Modules can import other modules.

If a module imports another module, functions and variables in the imported module are only available in the importing module, and are not propagated.

- All variables and functions of a module must have the module namespace. To hide variables and functions from other modules, use XQuery 3.0 `%private` annotations.
- XQuery modules can be stored as BLOB nodes or XML documents. BLOB nodes must contain the module in flat UTF-8 text, XML documents can have any encoding as long as it is correctly specified during the import. In XML documents, the string value of the document root element is used as the query. The query is the concatenation of all text nodes below that root node.
- XQuery modules that are stored outside of xDB are always expected to use UTF-8 encoding.
- xDB supports multiple locations per module URI. However, xDB does not support XQuery 3.0 features like forward references to global variables and allowing modules to reference each other without restriction. As a result, the order of the locations is of importance. For instance, the module content of the second location can reference items declared in the module content of the first location, but the module content of the first location cannot reference items declared in the module content of the second location.

Examples

The following example ignores the name of the `<queryModule>` root element.

```
<queryModule><![CDATA[ module namespace mns = 'http://some/namespace/uri';
    declare variable $mns:pi := 3.14159265;
    declare function mns:circle-area($r as xs:double) as
      xs:double { $r * $r * $mns:pi }; ]]>
</queryModule>
```

Use a CDATA block for the contents of the module. Otherwise embedded direct element constructors are interpreted as XML syntax, as described in the following code example:

```
<queryModule>
  module namespace foo = 'bar';
  declare variable $foo:element := <element>"Hello World!"</element>;
</queryModule>
```

The `$foo:element` variable contains the string "Hello World!", because `<element/>` was not escaped.

XQuery XML Schema support

xDB supports XML Schema within XQuery as follows:

- XML documents that have been parsed with schema validation and the `xhive-psvi` option enabled, expose the appropriate types in XQuery. When XQuery accesses the typed value of a node, for example by using the `data()` function, the resulting values are typed values. Typed values are `xs:integer` values, instead of `xs:untypedAtomic` values.

- Schemas can be imported using the **import schema** construct. The processor searches the catalog of the initial context item for a matching schema. The processor first uses any location hints, and then falls back to the namespace URI. DTDs are not supported.

A schema can be imported into queries and used to validate documents or XML fragments. Types declared in a schema can be used in XQuery type annotations.

XQuery Update Syntax

xDB implements the W3C XQuery Update Facility standard (<http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/>).

XQuery updates are evaluated using snapshot semantics: the query is evaluated completely before the updates are applied, so that update effects are not visible from within the query. This makes updates less error-prone, and it allows lazy evaluation and out of order execution.

To ensure that all updates have been generated and applied after execution, an XQuery using the update syntax is not evaluated lazily but at once. The results are cached. This caching can lead to increased memory usage if a query executes updates and returns numerous values.

For some operations on documents and libraries, xDB supports a [proprietary update syntax, page 211](#).

Proprietary XQuery Update Syntax

Some operations on documents and libraries are possible only with xDB's proprietary update syntax. For all other operations, the official XQuery Update Syntax should be used. The following xDB update syntax functions are available:

- **xhive:create-library(\$uri as xs:string) as empty-sequence()**

This function creates a library with the specified \$uri location. Any non-existent parent libraries in the path will be created as well.

- **xhive:insert-into(\$where as node(), \$what as item(*) as empty-sequence()**
xhive:insert-into-as-first(\$where as node(), \$what as item(*) as empty-sequence()
xhive:insert-into-as-last(\$where as node(), \$what as item(*) as empty-sequence()
xhive:insert-before(\$where as node(), \$what as item(*) as empty-sequence()
xhive:insert-after(\$where as node(), \$what as item(*) as empty-sequence()

These functions insert the \$what items relative to \$where items. The insert-into and insert-into-as-last behave identically. Atomic values within the \$what are converted into text nodes like they are in element constructors. If \$where is empty or not a node, an error is raised.

- **xhive:insert-document(\$uri as xs:string, \$document as document-node()) as empty-sequence()**
This function inserts the specified \$document at the \$uri location. If a document already exists there, an error is raised.
- **xhive:remove(\$nodes as node(*) as empty-sequence()**
xhive:delete(\$nodes as node(*) as empty-sequence()

These functions remove the specified \$nodes from their parents.

- **xhive:remove-library(\$uri as xs:string) as empty-sequence()**

This function removes the library at \$uri location, including all its children.

- **xhive:rename-to(\$what as node(), \$newName as xs:QName) as empty-sequence()**

This function renames the specified node to the \$newName value. This function raises an error if the target is not an attribute node, an element node, a processing instruction, or a document node. Processing instructions can only be renamed to unqualified local names, such as QNames without a namespace URI. To construct a QName, use the standard fn:QName(\$uri as xs:string?, \$qname as xs:string) as xs:QName function.

- **xhive:replace-value-of(\$where as node(), \$newContents as item(*) as empty-sequence()**

This function removes all children of \$where and replaces them \$newContents. This function is similar to the xhive:delete(\$where/node()),xhive:insert-into(\$where, \$newContents).

- **xhive:move(\$target as node(), \$sources as node(*) as empty-sequence()**

xhive:move(\$target as node(), \$anchor as node()?, \$sources as node(*) as empty-sequence()

These functions directly move DOM nodes into a new target. By default, they insert \$sources as last into \$target. If \$anchor is specified and not empty, the \$sources are inserted before \$anchor.

Moving has a potential performance advantage over removing/inserting nodes: if the \$where and \$newContents values belong to the same document, nodes need not be copied or imported.

Nodes covered by indexes with UNIQUE_KEYS flags can be moved. If any of the \$node child nodes use a unique index, moving elements with a delete node \$node and an insert node \$node into \$target statement generates a DUPLICATE_KEY exception. Using xhive:move(\$target, \$node) instead works.

Example

```
for $book in doc('bib.xml')/bib/book
    where $book/@year < 1990
    return
        xhive:remove($book)

for $book in doc('bib.xml')/bib/book,
    $review in doc('http://example.com/reviews.xml')//review
    where $review/@isbn = $book/@isbn
    return
        xhive:insert-into($book, $review)

xhive:insert-doc('/lib/newfile.xml',
    document {
        <root>
            ...
        </root>
    }
)
```

Data model differences

The xDB data model, which is the Document Object Model with extensions such as libraries, does not fit the XQuery/XPath data model perfectly. xDB follows the [Document Object Model \(DOM\) Level 3 XPath Specification](#) as follows:

- Entity reference nodes are treated as if they had been expanded. Queries never return entity reference nodes. Children of an entity reference node are treated as siblings of the entity reference node.
- Adjacent text and CDATA section DOM nodes are treated as single XQuery text nodes. The string value of the XQuery text node is the concatenation of the contents of the adjacent DOM text and CDATA section nodes. A query such as `//text()` returns only the first of each set of adjacent DOM nodes.
- BLOB nodes and library nodes do not have a representation in XQuery/XPath and are invisible to queries. However, selecting a library using the `doc()` function returns all elements in the library.

Additional XQuery namespace declarations

xDB provides several bound namespace prefixes by default. These namespaces can be overridden in the query prologue. In addition to the `xml`, `xs`, `xsd`, `xsi`, and `local` namespaces specified in the XQuery working draft, the following two prefixes have been predefined:

- `fn` is bound to the <http://www.w3.org/2003/05/xpath-functions> XQuery functions namespace. This namespace is also the default function namespace and allows using standard XQuery functions without prefix.
- `xhive` is bound to the <http://www.x-hive.com/2001/08/xquery-functions> xDB extension functions namespace.

More methods for XQuery

This chapter contains the following topics:

- [Use of type information in XQuery](#)
- [Parallel queries](#)
- [Using the XQuery Resolver](#)
- [Preparing XQueries](#)
- [Extending XQuery using Java](#)

Use of type information in XQuery

Applications can use typed information from indexes and PSVI in XQuery.

Samples

[TypedIndex.java](#)

Parallel queries

A particular subset of queries can be evaluated in parallel. For parallel evaluation, an executor instance must be provided using code like the following:

```
XhiveXQueryQueryIf query = ... ;
Executor executor = Executors.newCachedThreadPool();
query.setParallelExecutor(new AbstractXhiveXQueryExecutor() {

    @Override
    public Executor getExecutor() {
        return executor;
    }

}, XhiveXQueryQueryIf.XHIVE_PATH_EXPR);
XhiveXQueryResultIf result = query.execute();
while (result.hasNext()) {
    result.next();
}
result.close();
```

Parallel evaluation can improve performance: it can reduce the response time of queries, but there is some overhead involved that can reduce the total throughput.

There are two types of query parts that can be parallelized:

- Path expressions
- `fn:for-each`

If the `xhive:queryplan-debug` option has been turned on for the query, the output contains a message if the query is being parallelized.

Note: You have to call the `close()` method on the query result when the result items are no longer needed. This method terminates all background threads executing jobs in parallel mode.

The parallel executor must be a custom class implementing **XhiveXQueryExecutorIf**. The class **AbstractXhiveXQueryExecutor** provides an abstract implementation of the **XhiveXQueryExecutorIf** interface. Extend this class, rather than implementing the interface **XhiveXQueryExecutorIf**, so your code does not break if methods are added to the interface in future.

Interface **XhiveXQueryParallelJobIf** can be used to access sub-query information from within **ThreadPoolExecutor** hook functions **beforeExecute** and **afterExecute**.

Parallel Path Expression queries

If an FLWOR or path expression is evaluated on a library and no relevant indexes can be found, the query evaluation descends to the child libraries. The expression on each child library is evaluated separately. This step can be parallelized. The database creates jobs for the expression evaluation on each child library and submits them to the executor supplied by the user. Parallel query evaluation is most useful in cases where the searched child libraries are located on different disks, so the I/O load can be spread.

Generally, the expressions that can be parallelized are those that can use indexes, regardless of whether indexes are present or used. For examples of the kind of expressions that can be optimized, see [Value and element name indexes](#), page 189.

Parallel `fn:for-each` queries

The XQuery 3.0 function `fn:for-each($seq as item()*, $f as function(item()) as item()*) as item()*` applies the function `item $f` to every item from the sequence `$seq` returning the concatenation of the resulting sequences in order. When parallelizing this function, a configurable number of items of `$seq` is passed to function `$f` in its own thread.

Limitations

Parallel execution is more complex than non-parallel execution. If parallel execution does not improve performance it is recommended to run queries non-parallel. The following xquery constructs cannot be handled by parallel xquery parts:

- Variable declarations of function items.
- Variable declarations of other modules.

- Temporary nodes. Constructed elements and attributes are temporary nodes.
- Function items with fixed position arguments.

Samples

[ParallelPathExpressionQuery.java](#)

[ParallelForEachQuery.java](#)

Using the XQuery Resolver

xDB's XQuery engine offers applications fine-grained control over the document resolution process through an XQuery resolver. The `XQueryResolverIf` interface allows applications to control:

- Document resolving triggered by `doc()` and `collection()`
- Resolution of module imports
- Resolution of schema imports

The class `AbstractXQueryResolver` provides an abstract implementation of the XQuery resolver interface. Extend this class, rather than implementing the interface `XQueryResolverIf`, so your code does not break if methods are added to the resolver interface in future.

Register the implementation with `XhiveXQueryCompilerIf.setResolver(XQueryResolverIf)` or `XhivePreparedQueryIf.setResolver(XQueryResolverIf)`. In the latter case, module and schema imports are resolved using the default mechanism, because they happen during the construction of a prepared query and the resolver is only invoked for documents.

Samples

[XQueryResolver.java](#)

[XQueryCompiler.java](#)

API documentation

[com.xhive.query.interfaces.AbstractXQueryResolver](#)

[com.xhive.query.interfaces.XQueryResolverIf](#)

[com.xhive.query.interfaces.XQueryCompilerIf](#)

[com.xhive.query.interfaces.XhivePreparedQueryIf](#)

Preparing XQueries

The XQuery Compiler in the `XhiveXQueryCompilerIf` interface allows creating prepared queries. XQueries can be parsed once and used many times. The XQuery compiler also sets common options for all XQueries, such as available namespaces, options, commonly used functions, or modules. Using the same namespace prefixes or options for several queries reduces the amount of XQuery code.

The `XhivePreparedQueryIf` interface represents prepared XQueries. Prepared queries are thread safe and can be used in parallel, either by first creating an `XhiveXQueryQueryIf` object or by executing them directly.

For more information, see the `XQueryCompiler.java` sample code. The code prepares a query using an XQuery compiler, with an additional namespace prefix set, and runs the same XQuery using multiple threads.

Samples

[XQueryCompiler.java](#)

Extending XQuery using Java

xDB provides three extension mechanisms to integrate custom Java code with the XQuery engine. Extension functions can be declared in XQuery and assigned using the `XhiveXQueryQueryIf.setFunction(...)` function.

XQuery code can directly specify Java modules, for example

```
import module namespace math = "java:java.lang.Math";
math:sqrt(4), $math:E
```

All public methods and public static fields from the class are made available to the query. They are accessible using plain Java names and a translated version where all characters are lowercase and camel case is transformed into a hyphenated version. The hyphen is inserted between lowercase and uppercase characters, for example `getFoo()` is changed to `get-foo()` in XQuery, and `localURI` is changed to `local-uri`.

Java objects and instance methods

Parameters with an unrecognized type are returned as Java objects to XQuery. These objects can be passed to other Java functions, but all XQuery expressions fail.

In XQuery, Java values are used to call instance methods, as opposed to static methods. If a method is non-static, the instance it calls is passed as an additional first parameter.

Examples

```
/* Java code */
public int foo(String bar) { ... }

(: XQuery code :)
import module namespace eg = 'java:mypackage.Eg';
let $x := eg:new()
return eg:foo($x, 'param1')
```

Instances can be created using a constructor with the `eg:new(...)` syntax or injected from the outside as an external parameter.

```
import module namespace eg = 'java:mypackage.Eg';
declare variable $x external;
eg:foo($x, 'param1');
```

Type checking

XQuery parameters are checked for the correct type and promoted to Java objects according the table.

```
public static String foo(String bar, int baz, Iterator<XhiveNodeIf> nodes) { ... }
(: legal call :)
eg:foo("bar", 5, <element/>)
(: wrong type :)
eg:foo("bar", "baz", ())
```

The return value of the function is transformed to XQuery values exactly as in the `xhive:java()` method. It is possible to return Iterators, Collections, Arrays, and Sets.

Limitations

Two Java methods with different type parameter types can have the same name. In XQuery, functions with the same name are only allowed if they have a different number of parameters. In xDB, the query parser analyzes the input types from the query and tries to select the correct Java method accordingly. The parser calculates a score for each method based on how good the XQuery parameter types match the Java parameters. An error is reported if the more than one method has the best score. To direct the parser on which method to use, users can add **treat as** or **cast as** statements to the call, for example:

```
eg:foo(/some/path treat as element(*, xs:integer))
```


Catalogs and Validation

This chapter contains the following topics:

- **xDB catalogs**
- **Adding models to a catalog**
- **Linking models to documents**
- **Validated parsing**
- **Catalog methods**
- **Validating documents against models**
- **Post Validation Schema Infoset (PSVI)**
- **Accessing PSVI information**

xDB catalogs

xDB libraries store XML documents, sublibraries, and BLOBs. XML documents can be associated with a document type definition (DTD) or XML Schema to validate the document. DTDs and XML schemas are also referred to as *models*.

A catalog is linked to a library. By default, only the root library has a catalog where all models are stored. However, it is also possible to place a catalog in a sublibrary and split models over multiple catalogs. Catalogs in sublibraries are called *local catalogs*. Local catalogs override information in the root catalog and during queries the local catalog is searched first. If a root catalog and a local catalog contain a model with the same identifier, the model in the local catalog is used for all documents and descendants.

Identifying XML schemas and DTD models

Each model in a catalog has a unique identifier that depends on the schema type of the model:

- DTD models are identified by their public ID. If the DTD does not contain a public ID, xDB automatically generates an ID.
- XML schema models are identified by their filename.

Adding models to a catalog

DTDs and XML schemas are stored as ASModel objects. The ASModel interface is part of the DOM Abstract Schema specification and provides several interfaces for accessing model information, guided document editing, parsing and serialization.

Documents that are associated with a DTD can have internal subsets and contain a document type declaration along with DTD declarations. When the document is parsed with validation that internal subset is stored and available as an ASModel. The internal subset is not stored in the catalog and is only available on the document itself.

A document with a DTD can only have one external ASModel object. A document with an XML schema can have multiple ASModel objects since each schema document uses a separate ASModel.

A catalog can have a default DTD that is used during parsing. It is not possible to set a default XML Schema for a catalog.

Models can be added to a catalog in two ways:

- By parsing and validating a document. During validated parsing, the model linked to the document is automatically stored in the catalog.
- By parsing a model.

Linking models to documents

When accessing the schema information of a document, xDB automatically tries to locate the correct model in the library catalog where the document is stored.

Linking DTDs

Generally, a document that contains a `<!DOCTYPE>` declaration is linked to a DTD. The `<!DOCTYPE>` declaration specifies a public ID and a system ID, like the following:

```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "svg10.dtd">
```

Where `-//W3C//DTD SVG 1.0//EN` is the public ID and `svg10.dtd` is the system ID.

When retrieving the active ASModel of a document, the system looks up the ASModel Id in all catalogs up to the root library. The active ASModel is set using an abstract schema. If a document does not contain a `<!DOCTYPE>` declaration, the system automatically adds a `<!DOCTYPE>` declaration with the ASModel Id, linking the document to the ASModel. The model can be replaced by adding a new model to the catalog and changing the Id to the new model.

Linking to XML schemas

A document can be linked to an XML schema in two ways:

- Using the schema location attribute in the document element, like the following:

```
<personnel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation='personal.xsd'>
```

- Using the schema-location parameter in the document configuration to normalize the XML document.

A document with XML schema can have more than one active model attached.

After document validation or validated parsing, the string concatenation of the ASModel IDs is stored in the xhive-schema-ids parameter of the document. The IDs are used to identify the ASModels for validation or access to the PSVI interfaces.

Validated parsing

If a document is parsed with validation and the models are not found, the models are automatically stored in the catalog. If a document is parsed without validation, the models are not stored in the catalog. xDB includes DOM configuration options that allow validation without storing the model, or storing the model and internal subsets without validating the document.

ASModel resolutions are handled separately for DTD and XML schema models.

DTDs

When a document contains a `<!DOCTYPE>` declaration and the document is parsed with validation, xDB attempts to locate the DTD, as follows:

- If the `<!DOCTYPE>` declaration specifies a public ID, the public ID is used to identify the ASModel.
- If the `<!DOCTYPE>` declaration does not specify a public ID or a DTD matching the public ID does not exist, the system uses the default DTD.
- If no default DTD is specified, the system ID specified in the `<!DOCTYPE>` declaration is used to locate a DTD in the file system.

The DTD is stored in the closest local catalog, either under the specified public ID or a public ID generated by xDB.

Note: If the `<!DOCTYPE>` declaration does not specify a public ID, xDB stores a DTD for each document that is parsed with validation, even if they point to a DTD with the same system ID. Loading the DTD in advance and using the resulting ASModel as the default prevents storing a DTD for each document.-

XML schema

When parsing with validation, XML schema models are identified, as follows:

- If the document is parsed using the LSParser interface and a schema-location parameter is specified in the LSParser configurations settings, xDB validates against the defined ASModel. The schema-location parameter is added to the document.
- If the name space declaration of the document contains a `noNamespaceSchemaLocation` or `schemaLocation` XML schema instance attribute, xDB uses the corresponding ASModel for validation.

Note: If two models have the same target namespace, using the schema-location configuration parameter to define a model overrides using the schema location attributes.

Catalog methods

A catalog is a special library for storing schema documents, which are represented by ASModels from the W3C abstract schema specification. By default, only the root-library has a catalog, but you can set a local catalog by calling **addLocalCatalog()** on a library. When that is done, catalog operations on that library or its descendants will first call on this library.

A document with XML schema can have more than one active model attached.

Linked models can be changed by setting the schema-location parameter in the document configuration settings or by using the `setActiveASModel` and `addAS` abstract schema functions. These functions also modify the schema-location parameter value.

API documentation

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

[com.xhive.dom.interfaces.XhiveCatalogIf](#)

[org.w3c.dom.as.ASModel](#)

Validating documents against models

A document stored in xDB can be validated against its model. Documents using an XML schema are validated differently than documents using a DTD.

DTDs

The DTD validation process uses the ASModel defined by the public ID of the document. Validation fails if a DTD with that public does not exist in the catalog. The [ASModel](#), [page 126](#) interfaces contain the methods for validating documents that use a DTD.

XML schema

The configuration settings of the document determine the XML schema validation process, as described in [Normalizing XML documents](#), [page 102](#). The validation process attempts to locate the ASModels associated with the document, then validates the document against the model. The location of the model is specified either in the schema-location parameter in the configuration settings, or in the `noNamespaceSchemaLocation` or `schemaLocation` attribute of the document.

If two models have the same target namespace, using the schema-location configuration parameter to define a model overrides using the schema location attributes.

Post Validation Schema Infoset (PSVI)

xDB supports the Xerces XML Schema API that provides access to the Post Schema Validation Infoset (PSVI) and XML schema model information. This API has not been submitted to W3C, yet. The interfaces can change in the future.

The XML Schema API can be used to traverse XML schema components like type definitions, element declarations, and schema constraints. PSVI information, such as validity, validation context, normalized value, type definition, and member type definition can be accessed for individual nodes.

Nodes storing state information need more disk space. Users can set a configuration option to enable PSVI information storage. If this option is not set, queries do not support data types and the XML Schema API is only partially accessible. For example, node validity information is not available.

When using the XML Schema API to access schema information, the `xhive-schema-ids` attribute value identifies the corresponding ASModels. This value specifies the schema IDs corresponding to information stored by the configuration `schema-location` parameter and `schema-location` attributes. If possible, the `xhive-schema-ids` value excludes the schema locations of the attributes that the `schema-location` value overrules.

Related topics

[XML Schema data types](#)

[XML Schema model information](#)

[Xerces XML Schema API](#)

[Accessing PSVI information](#)

[DOM configuration](#)

Accessing PSVI information

Schema and validation information of attributes and elements can be accessed using the Xerces XML Schema API interfaces. To access the PSVI information of an element, the element must be cast to an **ElementPSVI** object. The following code example describes how to retrieve the validity of a node:

```
Element email = (Element) document.getElementsByTagNameNS(null, "email").item(0);
ElementPSVI elemPsvi = (ElementPSVI) email;
short validity = elemPsvi.getValidity();
```

Schema information is traversed by retrieving element declarations, attribute declarations and type definitions. The following code example describes how to retrieve the data type name of an element:

```
XSTypeDefinition elemTypeDef = elemPsvi.getTypeDefinition();
String typeName = elemTypeDef.getName();
```

Using the DOM level 3 **TypeInfo** object is another way to access data type information of elements and attributes. The **TypeInfo** is part of the **XhiveNodeIf** interface.

The "xhive-psvi" parameter of the `LSParser` object must be enabled during parsing to access the entire PSVI information. A document can be created using the `createDocumentPSVI` method in the **XhiveLibraryIf** interface.

Samples

[PSVI.java](#)

API documentation

[org.w3c.dom.TypeInfo](#)

[com.xhive.dom.interfaces.XhiveNodeI](#)

[org.apache.xerces.xml](#)

Administering xDB

This chapter contains the following topics:

- **Admin Client**
- **Web client**
- **Using the command-line client**
- **Creating and restoring backups**
- **Using the backup() method**
- **Using the restoreFederation() method**
- **Using the library backup() method**
- **Using the restoreLibrary() method**
- **Connecting to a federation backup**
- **Managing detachable libraries**
- **Duplicated transaction log files**
- **Monitoring statistics**
- **RAM segments**
- **Read-only federations**
- **Federation sets**
- **Using Secure Sockets Layer (SSL)**
- **Checking database consistency**
- **Message logging**
- **Message logging areas**
- **Message logging framework**

Admin Client

The xDB Admin Client (also known as Administration Client or administration tool) provides developers, administrators and superusers access to xDB functions through a Swing-based graphical user interface. The xDB distribution includes the Admin Client java source code, allowing developers to review its use of the xDB API to perform its tasks.

Admin Client features include:

- a menu-based GUI with menu bar, context-sensitive right-click menus and toolbars

- a data browser that displays database contents using an explorer-like tree view with tabbed content panels, popup dialogs and messaging
- server, federation, database and segment management
- consistency checking
- backup and restore
- data import and export
- data serialization and deserialization
- user and group management
- library and document management
- index management
- XQuery and XPointer execution and debugging

Using the xDB Admin Client

To start the xDB Admin Client:

1. Execute the **xdb admin** command in the `\bin` directory of the xDB installation. On Windows platforms, you can also use the Windows start menu.

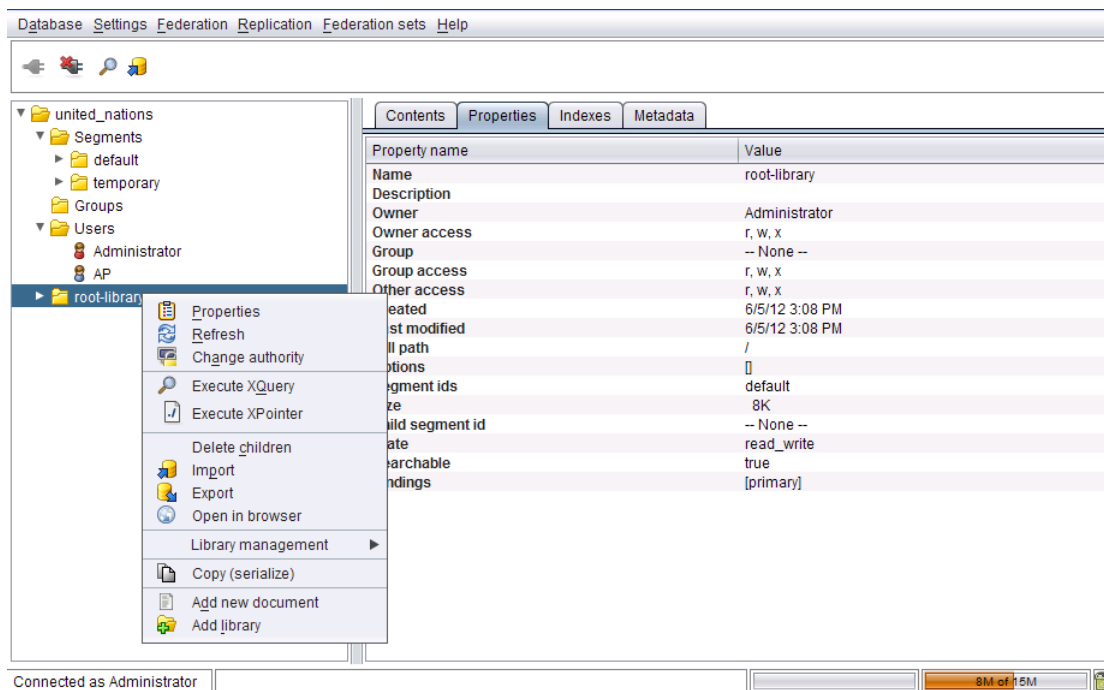
An alternative way to run the Admin Client is to execute the **xhive-ant run-admin** command in the `\bin` directory of the xDB installation. This command compiles and runs the client from the included source files.

2. Select **Database > Connect**.
The Connect to a database window displays.
3. Enter a valid database name, user name, and password, then click OK.

The Admin Client displays a tree view of the database in the left panel. The panel to the right displays details of the node selected in the tree view. Many object-specific operations are accessible by right-clicking on items in either panel.

The menu bar provides access to general functions. You can use the Help About option to determine which xDB version you are running.

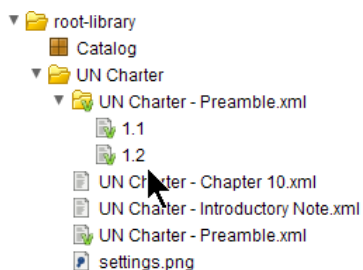
The status bar in the bottom right corner is a memory usage indicator. Clicking the trash can icon next to it forces a garbage collect.

Figure 5 Admin Client tree view with the context menu of the root library

There are up to four main nodes in the database tree:

- The **Segments** node provides the database Administrator user access to the segment properties and file paths of the database.
- The **Groups** and **Users** nodes are used for viewing, adding, and removing users and groups.
- The **root-library** is the top-level library that contains all other libraries and documents in a hierarchical order.

The tree can also contain special types of folders, such as Catalogs and Version folders. Catalog folders contain DTDs and XML schemas. Version folders contain versioning information for versioned documents.

Figure 6 Library hierarchy example

Most functions are accessible by right-clicking an item in the tree view or in the Contents tab, and selecting the corresponding option from the corresponding popup-menu. For example, you can selectively **Refresh** a part of the tree view, and you can query, import or export the contents of a specific library, and add or delete users and groups.

Additional functions are available in the main menu, for example for:

- creating and deleting databases
- changing passwords
- entering a new license key
- performing backup and restore
- checking federation consistency
- using replication

Note: Admin Client preferences, including the last query executed and the last database connection, are automatically stored using `java.util.prefs`.

Creating a database using the Admin Client

To create a database using the Admin Client:

1. Select **Database > Create database** from the Admin Client menu bar.
2. In the Create database dialog, enter a database name, a valid superuser password, and a database administrator password.

3. Enter optional parameters, if required. The **Path** option allows you to specify the location of the default file of the default segment, either as an absolute path, or as a path relative to the default database location of the federation. The **Max size** option specifies the maximum size (in bytes) that the database file is allowed to grow to. A value of 0 means the size is unlimited.
4. If you want to use a Custom configuration, enter the path to your [configuration file, page 230](#).
5. Click **OK**.

To connect to the new database, select **Database > Create database** from the Admin Client menu bar. In the **Connect to database** dialog, choose the **Database name** you want to connect to, and enter a valid username and password.

Database configuration files

When a database is created, a database configuration file is used to determine what segments and segment files to create.

Note: Database configuration files do not affect existing databases.

The database configuration file is in XML format, and can include the following elements:

- `<xhive-clustering>`, [page 231](#)

- [<segment/>](#), page 231
- [<file/>](#), page 232

The example configuration file below defines a default database:

```
<xhive-clustering>
  <segment id="default"/>
</xhive-clustering>
```

<xhive-clustering/>

This document element can be used in a database configuration file.

Child elements

The <xhive-clustering/> element can have the following child elements:

- [<segment/>](#), page 231

<segment/>

The <segment/> element creates a database segment.

Attributes

Name	Default	Description
id	-	The ID for the segment. Required attribute. The segment ID should be unique within the configuration file.
path	-	The path to the location of the default database file. Optional attribute. If the path is not supplied, the default database file is stored in the same directory as the default file of the federated database.
max-size	0	The maximum number of bytes for the default file. By default, the max-size attribute value is 0, which means the file can have an unlimited number of bytes. Optional attribute.
temporary	false	Specifies whether this segment is a temporary data segment. Optional attribute.

Child elements

The <segment/> element can have the following child element:

- [<file/>](#), page 232

<file/>

The <file/> element creates a database file.

Attributes

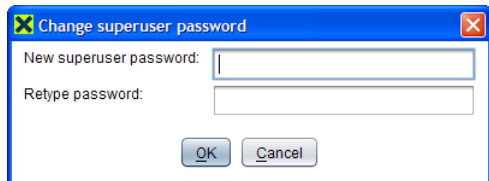
Name	Default	Description
path	-	The path to the location of the default database file. Optional attribute. If the path is not supplied, the default database file is stored in the same directory as the default file of the federated database.
max-size	0	The maximum number of bytes for the default file. By default, the max-size attribute value is 0, which means the file can have an unlimited number of bytes. Optional attribute.

Changing the superuser password using the Admin Client

To change the superuser password of the current federation with the Admin Client:

1. Select **Federation > Change superuser password** from the Admin Client menu bar.

The Change superuser password dialog opens.



2. Enter the new password in the field **New superuser password**.
3. Retype the new password in the field **Retype password**.
4. Click **OK**.
5. If required, enter the current **Superuser password**.

Changing the administrator password using the Admin Client

To change a database administrator password with the Admin Client:

1. Select **Database > Reset admin password** from the Admin Client menu bar.

The Reset admin password dialog opens.



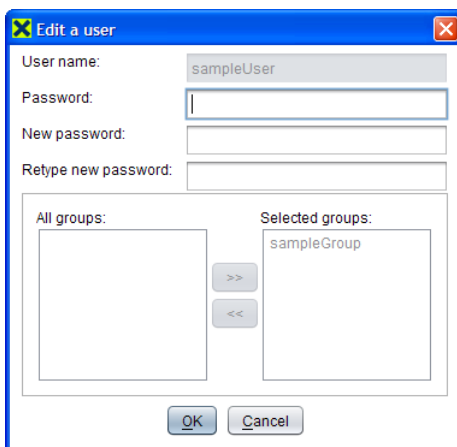
2. Select the **Database name** of the database on which you want to reset the administrator password.
3. Enter the **Superuser password** of the current federation.
4. Enter the new administrator password in the field **Administrator password**.
5. Retype the new administrator password in the field **Retype password**.
6. Click **OK**.

Changing a user password using the Admin Client

To change a user password with the Admin Client:

1. Connect the Admin Client to the database.
2. Navigate to the **Users** list.
3. Right-click the name of the user.
4. Select **Properties** from the right-click menu.

The Edit a user dialog opens, with the selected user name filled in. This dialog also shows the group(s) that the user belongs to.



5. Enter the current password.

The Administrator can assign groups, and can set a new password for another user without providing the current password.

6. Enter the **New password** for the currently selected user.
7. Retype the new administrator password in the field **Retype new password**.
8. Click **OK**.

Importing data

The **Import** option in the right-click menu of library nodes in the Admin Client opens a dialog for importing data from documents and/or directories into the selected library.

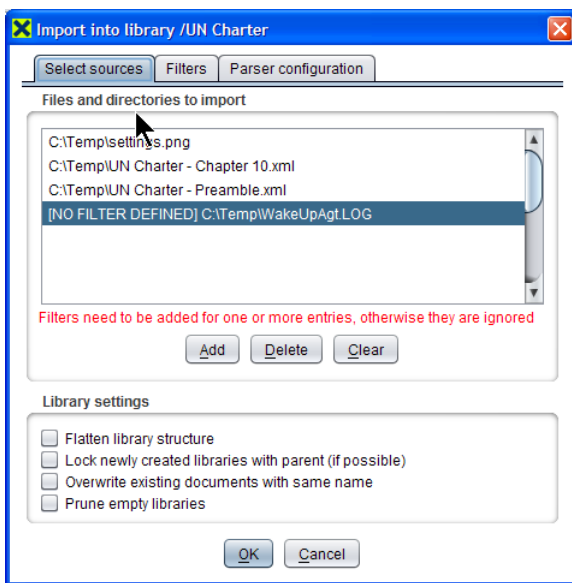
By default, xDB recreates the directory structure of an imported library.

To import data into a library:

1. Right-click on a library node and select **Import**.

The **Import into library** dialog appears, with the **Select sources** tab selected.

2. Use the **Add**, **Delete** and **Clear** buttons of the **Import into library** dialog to assemble a list of files and/or directories to import.



The **Add** button opens a dialog for selecting items to add to Files and directories to import.

3. Select **Library settings** as required:

Library settings	Description
Flatten library structure	Select this option to store all imported data directly in the target library.
Lock newly created libraries with parent	Select this option to lock imported libraries with the parent library.
Overwrite existing documents with same name	Select this option to overwrite documents with identical names if they already exist in the library.
Prune empty libraries	Select this option to ignore empty libraries.

4. Click the **Filters** tab and define import filters for all file types to import.

This tab contains user-definable filters that determine the storage type for each import file type. Filter definitions are stored in the Admin Client preferences. By default, the importer includes

filters for files with .xml and .xsl extensions. Files with file extensions that have no filter defined are ignored and not imported into the database.

If you want to import graphics from files of file type .gif, you should add a file filter for that type, with storage type **Blob**.

5. Click the **Parser configuration** tab and specify the parser options.

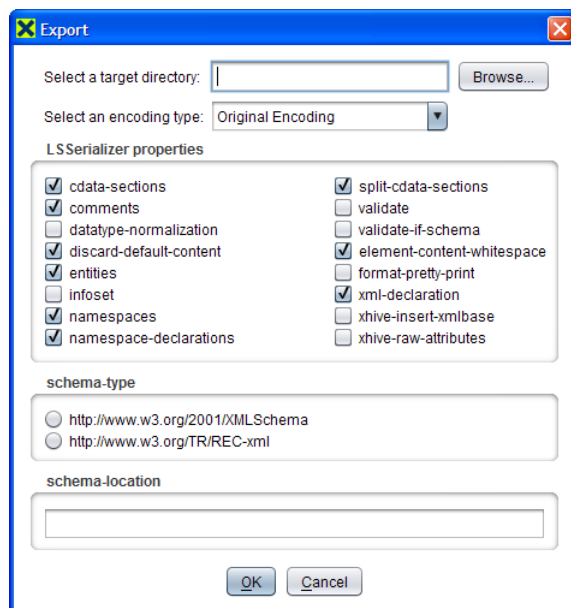
You can specify whether to use validation, what information items of the original file to preserve in the parsed Document, and other properties. The parser configuration options are not stored in the Admin Client preferences.

Exporting data

Data can be exported using the **Export** option in the right-click menu of library, BLOB, and document nodes. You can select export location and DOM configuration options.

To export data:

1. Right-click on the library, BLOB, or document node that you want to export.
2. Select option **Export** from the right-click menu.
The Export dialog appears.



3. Select a target directory.
4. Configure export options as required.
5. Click **OK**.

Backing up a federation

You can back up a federation using the **Backup** option of the **Federation** menu. The **Backup** dialog has options to create either a normal, an incremental or a standalone backup. If the federation contains detachable libraries, they can be excluded from the backup.

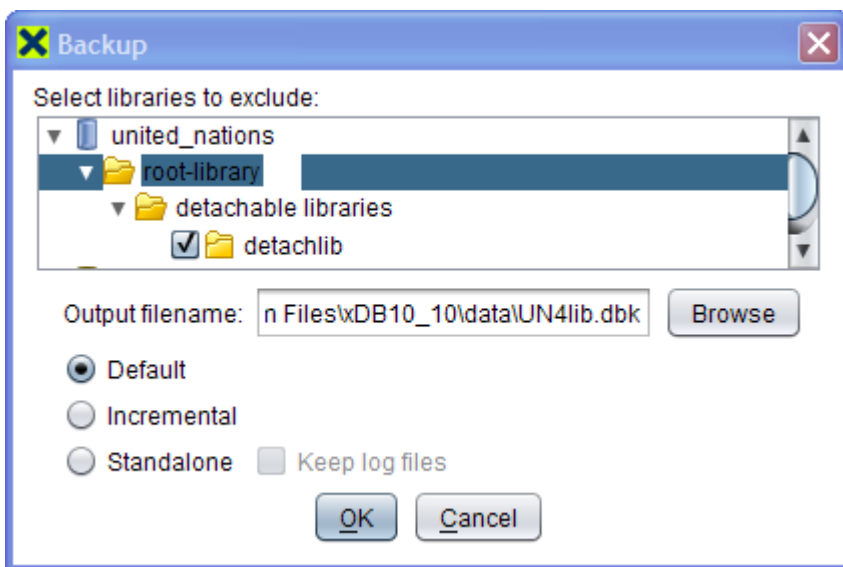
Note: It is good practice to use consistent, self-explanatory file names and file extensions for backups, and to keep related backup files together in a single, dedicated and secure location. For example,

keeping a backup and its subsequent increments in one place can make the restore process easier and more reliable. Creating backups and their increments only to new files in an empty location helps to prevent accidental overwriting of any preceding backups or increments. Preferably, a backup location contains *only* valid backup files. The [Admin Client, page 227](#) has a **Backup management** dialog that shows files name, backup type, creation date/time and other [backup metadata, page 266](#). This dialog can display a single backup file, or all backup files in an entire directory, optionally with its subdirectories, provided the selected directories contain *only* backup files.

To back up a federation:

1. Connect of the Admin Client to the federation that you want to back up.
2. Select option **Backup** from the **Federation**. If required, enter the superuser password. The **Backup** dialog appears.
3. Specify an output directory and filename.

Note: Choose the filename with care. If a file with the same name already exists, it will be overwritten without warning.
4. Select a backup option: **Default**, **Incremental** or **Standalone**. With the **Standalone** option, you can choose to keep log files.
5. Optionally, if the federation has detachable read-only libraries that you want to exclude from the backup, expand the tree view of the **Select libraries to exclude** control, and mark them for exclusion.



6. Click **OK**.

Restoring a federation backup

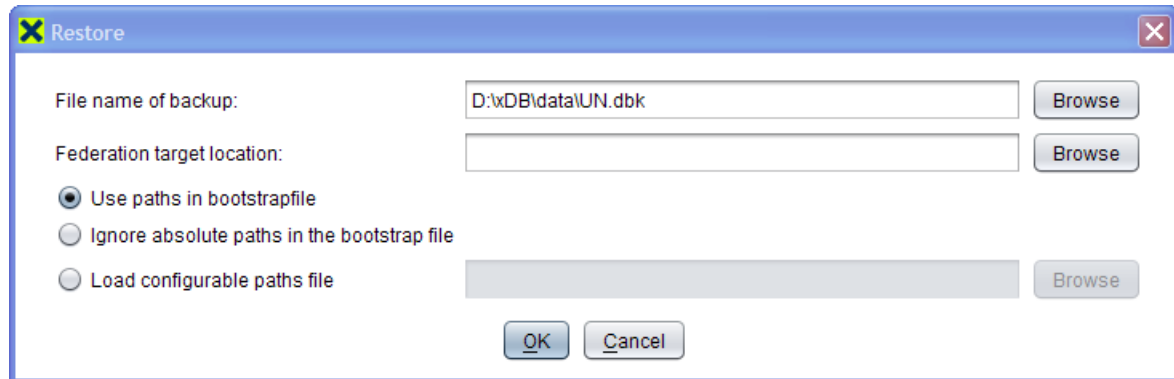
You can restore a federation from a backup file using the **Restore** option of the **Federation** menu.

Note: The Admin Client has a **Backup management** dialog that shows backup file name, backup type, creation date/time and other [backup metadata, page 266](#). This dialog can display a single backup file, or all backup files in an entire directory, optionally with its subdirectories, provided the selected directories contain *only* backup files.

To restore a federation:

1. Select option **Restore** from the **Federation** menu of the Admin Client. If required, enter the superuser password.

The **Restore** dialog appears:



2. Specify the federation backup file that you want to restore.
3. If you don't want to restore to the default location, specify a **Federation target location** for the restored federation.

Note: If you want to restore to an existing federation, ensure that there is no page server connected to it.
4. Select a restore **paths** option. Restore can use the paths in the bootstrap file, or ignore absolute paths in the bootstrap file, or load a file that maps new restore paths to the old paths. For an example of such a file, refer to the section about [the xdb restore command, page 264](#).
5. Click **OK**.
6. If you are restoring a set of incremental backups, repeat the steps above for each increment, in the order in which the backup files were created.

Depending on your choices and circumstances, the restore process may require further action. For example:

- If the specified federation target location already contains federation files, restore asks for permission before overwriting them. If you refuse permission, the restore will abort.
- If you try to restore to an existing federation while a page server is running on it, you'll get an error message. For restore to succeed, stop its page server.
- If you try to restore a backup increment to an existing federation, you will get an error message if its page server has been started between the previous and the current restore. Stop the page server, ensure that it stays down, and redo the entire restore, starting with the full federation backup.
- To verify the result, start the page server and run a consistency check on the federation.

Serializing data

You can serialize a library, document or BLOB to a file using the **Serialize** option in the right-click menu of the object. Subsequently, you can deserialize the object from the file to any library.

You can use serialization and deserialization for backup/restore of content, and for copying content from one library or database to another.

Note: Serialization files are not human-readable, and should not be edited.

To serialize data:

1. Right-click on the library, BLOB, or document node that you want to serialize.

- a. If the object is a library, select menu option Library management from the right-click menu.
2. Select option **Serialize**.
The Serialize dialog appears.
3. Select a target directory.
4. Enter a file name.
Use a meaningful file name, for example a name that describes the file content, or a name that relates to its purpose.
Note: If you use the name of an existing file, that file will be overwritten without warning.
5. Click **Serialize**.

Deserializing data

You can use the **Deserialize** option in the right-click Library management option of a library to add content from a file created using the **Serialize** option.

Note: Serialization files are not human-readable, and should not be edited.

To deserialize data:

1. Right-click on the library where you want to serialize data, and select submenu option **Library management**.
2. Select option **Deserialize**.
The Deserialize dialog appears.
3. Select a source directory and file.
4. Enter a file name.
Use a meaningful file name, for example a name that describes the file content, or a name that relates to its purpose.
Note: The file that you select must have been created using the **Serialize** option.
5. Click **Deserialize**.

Deserializing the root library

You can use the **Deserialize root library** option in the right-click menu of the database to replace the entire root library with content from a file created by serializing a library.

Note: Serialization files are not human-readable, and should not be edited.

To deserialize data:

1. Right-click on the database node in the Adminclient..
2. Select option **Deserialize root library**.
The Deserialize root library dialog appears.
3. Select a source directory and file.

Note: The file that you select must have been created using the **Serialize** option of the **Library management** menu of the root-library.

4. Click **Deserialize root library**.

Serializing users and groups

You can serialize the users and groups of a database using the **Serialize users and groups** option in the right-click menu of the database. Subsequently, you can deserialize the users and groups from the file.

You can use user and group serialization and deserialization for backup/restore of user data, and for copying user data from one database to another.

Note: Serialization files are not human-readable, and should not be edited.

To serialize users and groups:

1. Right-click on the database that you want to serialize.
2. Select option **Serialize users and groups**.
The Serialize users and groups dialog appears.
3. Select a target directory.
4. Enter a file name.

Use a meaningful file name, for example a name that describes the file content, or a name that relates to its purpose.

Note: If you use the name of an existing file, that file will be overwritten without warning.

5. Click **Serialize users and groups**.

Deserializing users and groups

You can use the **Deserialize users and groups** option in the right-click menu of the database to replace the users and groups with data from a file created by serializing users and groups.

Note: Serialization files are not human-readable, and should not be edited.

To deserialize data:

1. Right-click on the database node in the Adminclient.
2. Select option **Deserialize users and groups**.
The Deserialize users and groups dialog appears.
3. Select a source directory and file.

Note: The file that you select must have been created using the **Serialize users and groups** option.

4. Click **Deserialize users and groups**.

Comparison of online backup and serialization

xDB supports serialization and deserialization of individual libraries and documents with their metadata, such as indexing, versioning and authority information. The serialization and deserialization processes use an internal binary format. Some practical differences between online backup/restore and (de)serialization are described below.

Table 33 Comparison of online backup and serialization

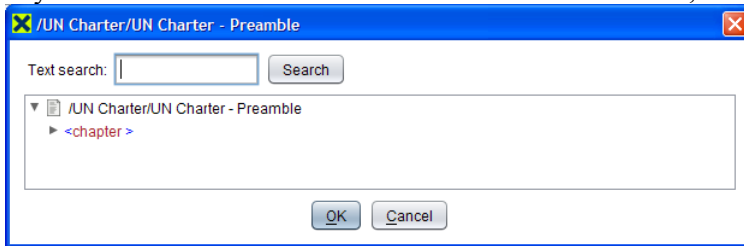
Online backup/restore	Serialization/Deserialization
Backs up a complete federation.	Stores individual libraries or documents.
Does not use locks or transactions.	Takes read locks on serialized data.
Fast.	Less fast.
Restores without running the server.	Deserializes with running server.
Restores exactly the same federation contents.	Allows deserialization to different federation, database or parent library.

Serialization and deserialization can be done using right-click menu options of Admin Client.

Editing documents

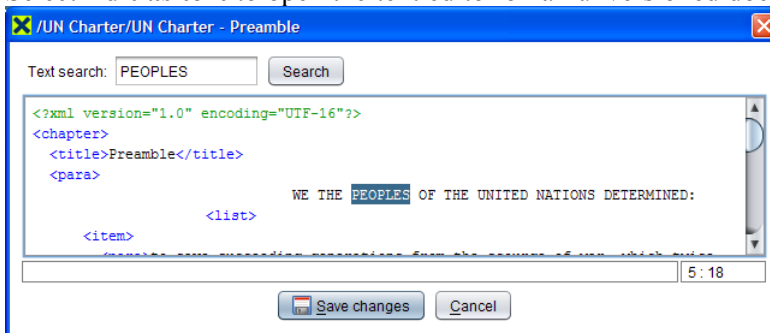
To edit documents in the xDB Adminclient:

1. Right-click on the document you want to edit.
2. The right-click menu offers a different editing option for versioned documents than for unversioned documents, because versioned documents are read-only.
 - If you want to browse an unversioned document as a tree, select **Browse document**.



In the document browser, you can right-click a node to open a menu with options to edit or delete the node.

- Select **Edit as text** to open the text editor on an unversioned document or document node:

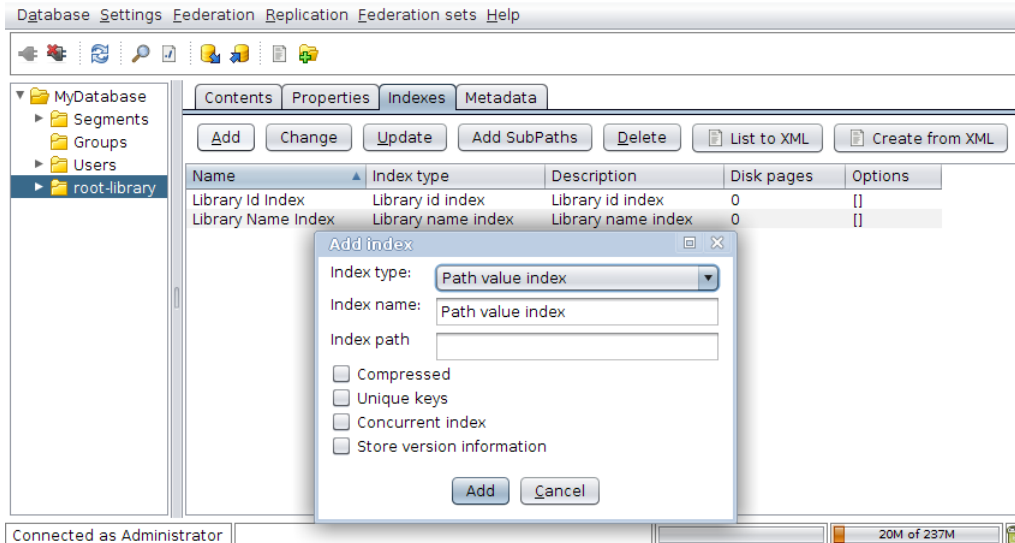


- Select **Checkout/Edit/Checkin** to open the text editor on a versioned document.
3. Edit the text.
 - To store your changes, select **Save changes**, specify parser options as required and click OK.
 - To discard your changes, click **Cancel**.

Adding indexes

When you select a library or a document in the left panel of the administration client, you can use the **Indexes** tab of the right panel to view and manage its list of indexes. For information on indexing, refer to [indexes](#), page 150.

Figure 7 Adding indexes



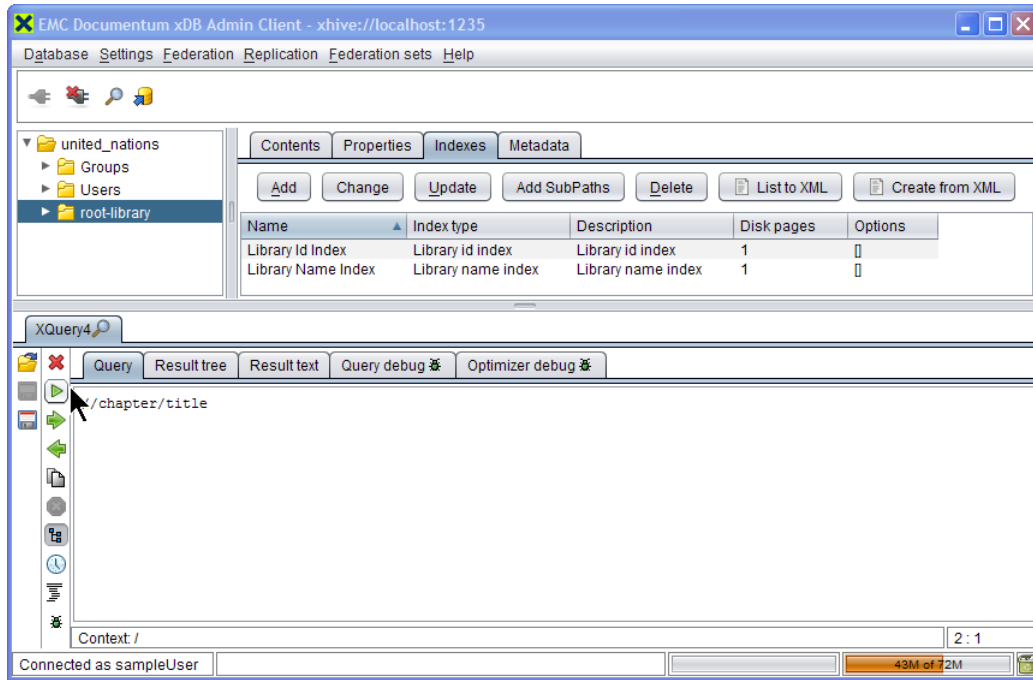
Merge tasks can be ran/killed by using the context menu

Running queries

XQuery is one of several querying mechanisms supported by xDB. XPath is a subset of XQuery - what you can do in XPath, you can do in XQuery, and more.

For libraries and documents, the **Execute XQuery** option in the right-click menu opens a new **XQuery** panel at the bottom of the window. Queries are executed in the context of the selected library or document. The Adminclient uses a read-only transaction (which does not take locks) for normal XQuery execution, but it will use a read-write transaction if it detects [XQuery update syntax](#), page 211 in your query.

Figure 8 XQuery tabs and options



The XQuery panel contains several tabs, as described in table (XQuery tab and options, page 242).

Table 34 XQuery tabs and options

XQuery tab	Description
Query	<p>On the Query tab, you can enter your XQuery text.</p> <p>Click the Run button (green triangle) on the toolbar at the left to run your query.</p> <p>The XQuery toolbar includes buttons to close the panel, to copy, format and debug the xquery, to load and save queries, and more. The last executed query is remembered.</p>
Result tree	<p>The Result tree tab displays query results in a tree view.</p> <p>If the query involves the original nodes that were not created through element constructors, they can be deleted from the tree and are deleted in the original database.</p> <p>A transaction is kept open while the tree is open, including old versions of the data. In a very active federation, the result tree tab should be disabled or cleared using the Do not use result tree button, which causes the query transaction to be closed immediately after completing a query. The query result can still be viewed in the Result text tab.</p> <p>The Result tree tab and button are disabled for update queries.</p>
Result text	<p>The Result text tab displays the query results as text.</p>

XQuery tab	Description
Query debug	<p>The Query debug tab displays debug information about the query, if debug options were configured in the query. For more information about debug options, see XQuery options, page 179.</p> <p>The following example shows debug output for a value index query:</p> <pre> Creating query plan on node /UN Charter for expression .../descendant-or-self::node()/child::chapter[@number < ...] Looking for value index on chapter/@number (sorted) (type=INTEGER) Found index "Value index" Looking up "3" in index "Value index" Using query plan: index(Value index) </pre>
Optimizer debug	<p>The Optimizer debug tab displays an index plan for a path expression. The output contains detailed information on the indexes that are considered, including those that are eliminated, and how a query plan is constructed.</p>

When executing an XPath/XUpdate/XQuery, if the results are idle, the underlying session will timeout. The user will then be given a choice whether or not to re-execute the action: if not then the results tree will be disabled. The default timeout is set to 5 minutes and can be set through the Options dialog of the Settings menu.

Profiling XQueries

Xquery profiling can be useful in finding why a query runs slow. The XQuery panel of the Admin Client provides an option for profiling XQueries. For information about the XQuery panel, see [Running queries, page 241](#).

Click the **Show Query plan** (clock) button of the XQuery panel to open a new window showing a tree view of the static query plan.

Figure 9 XQuery query plan dialog after profiling

Nodes	ms elapsed	elapsed %	# pages	pages %	# values
▼ XQueryQuery calls=1	22	100 %	5	100 %	1
▼ querytext	0	0 %	0	0 %	0
declare option xhive:index-debug 'true'; declare optio...	0	0 %	0	0 %	0
functions	0	0 %	0	0 %	0
variables	0	0 %	0	0 %	0
modules	0	0 %	0	0 %	0
▼ let calls=1 location=query:5:1 type=item()*	22	100 %	5	100 %	1
▼ path calls=1 location=query:5:21 numExpr=2	21	95 %	3	60 %	0
▼ indexplans	12	54 %	2	40 %	0
▼ indexplan context=/ node=primary	12	54 %	2	40 %	0
lookup calls=1 conditions=1 index=mp1	12	54 %	2	40 %	0
▼ path path=.../child::feed/child::docf. ftcontains	0	0 %	0	0 %	0
root calls=1 location=query:5:21	0	0 %	0	0 %	0
▼ let calls=1 location=query:5:54 type=item()*	22	100 %	5	100 %	1
▼ for calls=1 location=query:6:15 type=item()*	0	0 %	2	40 %	2
▼ path calls=1 location=query:6:28 numExpr=2	0	0 %	2	40 %	2
▼ indexplans	0	0 %	0	0 %	0
indexplan context=/ node=primary	0	0 %	0	0 %	0
indexplan context=/dewiki20m.xml	0	0 %	0	0 %	0
indexplan context=/bib.xml	0	0 %	0	0 %	0
▼ path	0	0 %	0	0 %	0
root calls=1 location=query:6:28	0	0 %	0	0 %	0
variable-access calls=2 location=query:6:71	0	0 %	0	0 %	2
▼ element-constructor calls=1 location=query:7:8	22	100 %	5	100 %	1

Click the **Profile** button at the top of the Query plan window to run the XQuery with profiling enabled. After profiling a query, the query plan dialog shows the values of performance-relevant attributes, with calculated percentages of the totals in additional columns.

You can copy the resulting information to clipboard as XML text by clicking the **Copy selection** button. If you first select a node or a range of lines, only the selected part of the display is copied to clipboard.

Format

Profiling produces an XML document contains the original query text and a tree of XML nodes that represent the functions, modules, variables and expressions of the XQuery.

Where applicable, these nodes have a `location` attribute that points to the filename, line, and column number of the source file where this expression was parsed from. Many expressions also have additional attributes like the variable name for a `for` clause, or the function name for a function. In the document, outer expressions that are XML parent nodes consume the results from their child nodes, input expressions (for example the input to a `for` clause) come before output expressions (like the `return` clause in a `for` clause).

Profiled expression nodes will have the attributes `accumulatedTime`, `calls`, `values`, and `pagesRead`. These represent the total time spent evaluating this expression, the number of times this expression was evaluated, the number of values this expression produced, and the number of database pages that had to be accessed for producing the result.

In addition to timings, the `<path/>` nodes representing path expressions will have an `<indexplans/>` child node that contains the different index plans chosen for different libraries, and within those plans a description of the lookup steps used to evaluate the path expressions.

Note: The profiling XML document format is a work in progress and subject to change without notice.

Note: The profile will not account for pages read on the server and not transferred to the client in a client server deployment. In practice, this means `pagesRead` will not include pages read in concurrent indexes and multipath indexes. **Note:** If an expression uses a variable, the time spent and the pages read to evaluate that variable will be accounted twice: once for the expression that binds the variable, and once for the first expression that uses the variable. This is due to the lazy evaluation nature of xDB's XQuery implementation. However, the time spent on a the parent expression relative to the variable-binding expression will typically be correct.

Example XQuery profile

```
<?xml version="1.0" encoding="UTF-16"?>
<queryplan end-time="2010-05-04T15:55:35.038Z" start-time="2010-05-04T15:55:35.026Z"
  xDB-version="xDB main@621581">
  <XQueryQuery accumulatedTime="10" calls="1" pagesRead="5" values="1">
    <querytext>declare option xhive:index-debug 'true';
declare option xhive:queryplan-debug 'true';
declare option xhive:pathexpr-debug 'true';
(: declare option xhive:ignore-indexes 'mpl';:)
let $othello docs := /feed/doc[. contains text 'Othello'],
  $books := for $book in /bib/book[author/last = 'Stevens'] return $book
return <res>{ $othello docs, $books }</res></querytext>
    <functions/>
    <variables/>
    <modules/>
    <let accumulatedTime="10" calls="1" location="query:5:1" pagesRead="5"
      type="item()*" values="1" variable="othello docs@0">
      <path accumulatedTime="6" calls="1" location="query:5:21" numExpr="2"
        onlyChildren="true" pagesRead="3" returnBlobs="false"
        usesNotOrOr="false" values="0">
        <indexplans>
          <indexplan context="/" node="primary">
            <lookup accumulatedTime="2" calls="1" conditions="1"
              index="mpl" lookup="server-side" pagesRead="2" type="11"
              values="0"/>
          </indexplan>
        </indexplans>
        <path path=".../child::feed/child::doc[. contains text Othello]">
          <root accumulatedTime="0" calls="1" location="query:5:21"
            pagesRead="0" values="0"/>
        </path>
      </path>
    </let>
    <let accumulatedTime="10" calls="1" location="query:5:54" pagesRead="5"
      type="item()*" values="1" variable="books@2">
      <for accumulatedTime="2" calls="1" location="query:6:15" pagesRead="2"
        type="item()*" values="2" variable="book@1">
        <path accumulatedTime="1" calls="1" location="query:6:28"
          numExpr="2" onlyChildren="true" pagesRead="2"
          returnBlobs="false" usesNotOrOr="false" values="2">
          <indexplans>
            <indexplan context="/" node="primary"/>
            <indexplan context="/dewiki20m.xml" node="primary"/>
            <indexplan context="/bib.xml" node="primary"/>
          </indexplans>
          <path path=".../child::bib/child::book
            [child::author/child::last[. = &quot;Stevens&quot;]]">
```

```

        <root accumulatedTime="0" calls="1" location="query:6:28"
            pagesRead="0" values="0"/>
    </path>
</path>
<variable-access accumulatedTime="1" calls="2" location="query:6:71"
    pagesRead="0" values="2" variable="book@1"/>
</for>
<element-creator accumulatedTime="10" calls="1" location="query:7:8"
    pagesRead="5" qname="res" values="1">
    <sequence accumulatedTime="8" calls="1" location="query:7:8"
        pagesRead="5" values="3">
        <sequence accumulatedTime="8" calls="1" location="query:7:27"
            pagesRead="5" values="2">
            <variable-access accumulatedTime="6" calls="1"
                location="query:7:16" pagesRead="3"
                values="0" variable="othello docs@0"/>
            <variable-access accumulatedTime="2" calls="1"
                location="query:7:30" pagesRead="2"
                values="2" variable="books@2"/>
        </sequence>
        <literal-content accumulatedTime="0" calls="1" location="query:0:-1"
            pagesRead="0" value="" values="1"/>
    </sequence>
</element-creator>
</let>
</let>
</XQueryQuery>
</queryplan>

```

Web client

The web client provides administrators and superusers access to some key database functions, including management of users and indexes and execution of queries. It is based on the xDB REST API.

By default, its server starts automatically as part of the database server and runs on port 1280. During xDB installation, this can be disabled or set to a different port. These settings are stored in the `xdb.properties` configuration file. When running the database server from the command line, you can choose to disable the web server.

To unlock administrative actions on the federation, the username `superuser` must be entered, together with the password provided during setup.

To connect to a database, the username `Administrator` must be entered, together with the password that was entered when creating the database.

Using the command-line client

xDB's command-line client offers commands for various administrative purposes, including data backup/restore. It can be used from terminals or scripting languages. For a list of commands, refer to [Command-line client commands, page 248](#).

The command-line client can execute single commands and can also run as an [interactive console](#), [page 247](#). Commands are always executed in auto-commit mode. All changes are made persistent after each command, before control is returned to the command line.

To use the command-line client to run a single **xdb** command, enter that command on your operating system's command line in the bin subdirectory of the xDB installation. **Note:** The Windows installer automatically adds the xdb command to the PATH variable.

Single commands must be entered using the following syntax:

```
xdb <command> [arguments]
```

Enter `xdb help` to display the available commands with their descriptions.

Enter `xdb help <command>` to display the available options for a command.

Parameters containing whitespace must be enclosed in single quotes (') or double quotes (") or escaped by preceding the whitespace with a backslash (\). Quotes within parameters can be escaped using the backslash character (\' or \"), the backslash itself is escaped by doubling (\\).

The xdb command accepts GNU-style options, either in a long version such as `--federation` or as an abbreviated version such as `-f` for some options. If an option takes a value, that value must directly follow the option, separated by whitespace, for example `--federation xhive://localhost:1235`. Abbreviated options can be clustered if only the last option takes a value, for example `-vyf xhive://localhost:1235`.

Most commands have a number of options in common; these are called [global options](#), [page 251](#). When passed to the xdb command, these options also serve as default values for any subsequent commands in interactive mode. The interactive console will cache username and password, server address, server port, web server properties and database name between commands, so after the first command that accesses a given database, subsequent commands need not ask for them again.

Default values for most command options are stored in the `xdb.properties` configuration file in the home directory of each xDB user. These options need not be specified on the command line every time a command is invoked. The default values in the configuration file can be modified by specifying the corresponding parameter on the command line. For more information about the configuration settings and locations, see [Configuration files](#), [page 68](#).

Starting the interactive console

The interactive console accepts all regular xdb commands, using the same syntax. To start the interactive console, run the xdb command without options:

```
xdb
```

To close the console, type **exit**.

Interactive console example

```
user@localhost: ~ $ xdb
xDB 10.5.0 command line client (c) 1999-2013 EMC Corporation
Type 'help' for a list of commands and options, type 'exit' to leave the shell.
xdb> ls -d united_nations
abc.xml
```

```
xdb> import myfile.xml /
Parsing XML document myfile.xml ...
Stored 1 file(s).
xdb> ls -d united_nations
abc.xml
myfile.xml
xdb>exit
```

Command-line client commands

Commands

The table below lists the commands that are available from the xDB [command-line client](#), page 246.

Table 35 Command line client commands

Command	Description
add-binding	bind a library to specified xDB server node in addition to all the existing bindings.
add-log-directory	add a secondary transaction log files directory to a node
add-node	add a server node
add-segment	add a segment to the database
add-file	add a file to a database segment
admin	start the admin client graphical user interface
attach-library	attach a library back into a database from detached state.
backup	backup a complete federation
backup-library	backup a library or multiple libraries within a database
cat	prints a file
cd	change the current directory
change-binding	bind a library to specified xDB server node.
check-database	check database consistency
check-federation	check federation consistency
check-librarychild	check library child consistency
check-node	check node consistency
clean-database	clean a database's MultiPath index segments
clean-library	clean a library's MultiPath index segments
configure-federation	change federation settings (superuser password, license key)
create-database	create a database
create-federation	create a federation
create-replica	create a replica

Command	Description
delete-database	delete a database
detach-library	detach a library if it is detachable
federation-set	manage a federation set (create, add federations, remove federations)
force-detach	forcibly detach a library on which the library is created.
help	print a help message
import	import documents into the database
info	print session information
ll	print a list of database objects and their attributes
ls	print a list of database objects
mkdir	create directories (always creates missing parents)
mv	move a file or folder
put	create a document in the database
remove-binding	remove specified server node as one of the binding nodes of a library.
remove-log-directory	remove a secondary transaction log files directory from a node
remove-node	remove a server node
remove-segment	delete a segment from the database
repair-blacklists	repair MultiPath index blacklists
repair-addblacknode	add black node to MultiPath index
repair-indexinfo	repair MultiPath index information
repair-mappings	repair MultiPath index file mappings
repair-merge	merge MultiPath index
repair-removeblacknode	remove black node from MultiPath index
repair-segments	repair MultiPath index segments
repair-set-usable	set MultiPath index usable
repair-searchable	flip library non-searchable flag (also repairs non-searchable flag issue from xDB 9.0.0)
repair-input-encoding	repair or check input encoding after upgrade to xDB 7.1
restore	restore a complete federation from a backup
restore-library	restore one library or multiple libraries within a database from a backup
rm	delete files/directories
run-server	start an xDB server
run-server-repair	Starts an xDB server node in repair mode, suitable for index repair.
set-file-maxsize	set max size of a segment datafile
set-library-state	set states of library
show-backup-metadata	show metadata of a specified backup file or multiple backup files in a specified folder. Metadata fields will be separated by tabs.
show-segment	show properties of a segment. Fields will be separated by tabs.
show-unusable-indexes	show information of all unusable indexes in specified database

Command	Description
show-unusable-libraries	show full paths of all unusable libraries in specified database
statistics	print statistics about MultiPath index
statistics-bl	print MultiPath index blacklists info
statistics-li	print subindex specific information
statistics-ls	print MultiPath index sub indexes info
stop-server	stop a running xDB server
suspend-diskwrites	suspend or resume disk writes
sync-entries	sync index record and index_info entries
update-node	update a server node
xquery	execute an XQuery, either given as a direct argument or through the --file option.

Creating a federation

When xDB is installed, a federation is created automatically. Additional federations can be created using the **xdb create-federation** command or the Admin Client.

The **xdb create-federation** command supports the following options:

Option	Description
-f --federation ARG	The absolute path and file name of the bootstrap file for the new federation.
--log ARG	Comma separated list of transaction log files directories for the new federation. The first one in the list represents the primary log directory. Relative paths, if used, are resolved relative to the bootstrap file directory. If not specified, a single log directory for the node will be created in default location. Note: For performance reasons, it is best practice to keep the transaction log files on a different physical hard disk than the database data pages.
--pagesize ARG	The database page size for the new federation.
-p --passwd <value>	The initial superuser password.

Example

The following example uses the **xdb create-federation** command to create a federation.

```
xdb create-federation --log /var/dblogs/fed1logs \  
  --federation /var/databases/Federation1.bootstrap \  
  --pagesize 8192 --passwd <password>
```

```
--passwd secret --pagesize 4096
```

Creating a database using the command line

1. Open a command line window and change to the bin directory of the xDB installation.
2. Run the **xdb create-database** command:

```
xdb create-database -p SuperUserPassword --adminpwd
AdminPassword DatabaseName
```

The xdb info command

The **xdb info** command sends debug information about open transactions and their associated locks to standard output.

Locks are shown as either R(ead) or W(rite) locks, with an internal ID and a short description of the locked object.

```
xdb> info -f xhive://localhost:1236
Session adminclient
Remote connection from Socket[...]
Connected as Administrator to united_nations
Joined by thread Thread[request handler Socket[...] [F:XhiveDatabase.bootstrap],5,main]
Transaction id = 3278
owns locks:
R 26 Document /CreateTransactionSample - UN Charter - Chapter 9
R c Index Library Id Index on /
R a Library /
R 1 Database united_nations
```

The **xdb info** command only shows open sessions. Closed sessions are not listed in any internal administration, and so are available for garbage collection if they are no longer referenced by user code.

A <page not in cache> entry in the description usually means that the locked object is new and its first page has not yet been entered in the database server cache. Another possibility is that the first object page has been removed from the cache to create space for other pages; in this case, the object name is not retrieved from disk, to avoid affecting the performance of current transactions.

Command-line client global options

Most xdb commands have the following options in common:

Table 37 Global command line options

Option	Description
-u --username ARG	The user name. If omitted, xDB can automatically use the superuser or administrator user name where needed.
-p --password ARG	The password of the user.

Option	Description
-f --federation ARG	The federation bootstrap path or URL.
-d --database ARG	The name of the database.
-c --cache ARG	The cache pages for the database session.
-y --non-interactive	Runs the non-interactive mode that does not ask for missing parameters.
--debug	Prints stack traces.
--stdout ARG	Redirects standard output to a file.
--stdout-append ARG	Redirects standard output to a file (append mode).
--stderr ARG	Redirects standard error output to a file.
--stderr-append ARG	Redirects standard error output to a file (append mode).
-v --verbose	Runs in extra verbose mode.
-V --version	Prints version information and exits.
-h --help	Prints overview of common options and commands.

Server-related commands

add-node

Adds an xDB server node.

Usage: xdb add-node [options]

Argument	Description
--logpath ARG	Comma separated list of transaction log files directories for the server node to be added. The first one in the list represents the primary log directory. Relative paths, if used, are resolved relative to the bootstrap file directory. If not specified, a single log directory for the node will be created in default location. If present, the log directories must be accessible to both primary and the specified non-primary node.
--port ARG	Port number of the server node to be added.
--host ARG	Host name of the server node to be added. Required.
--nodename ARG	Name of the server node to be added. Required.

add-log-directory

Adds a transaction log files directory to a server node.

Usage: xdb add-log-directory [options]

Argument	Description
--logpath ARG	Log directory to be added (relative or absolute path). Relative paths, if used, are resolved relative to the bootstrap file directory. It must be accessible to both primary and the (possibly) specified non-primary node. Required.
--nodename ARG	The name of the node to which the log directory will be added. Optional. Primary node by default.

remove-log-directory

Removes a transaction log files directory from a server node.

Usage: xdb remove-log-directory [options]

Argument	Description
--logpath ARG	Log directory to remove from the server node (relative or absolute path). Relative paths, if used, are resolved relative to the bootstrap file directory. The primary log directory cannot be removed. Required.
--nodename ARG	The name of the node from which the log directory will be removed. Optional. Primary node by default.

remove-node

Removes an existing xDB server node.

Usage: xdb remove-node [options]

Argument	Description
--nodename ARG	Name of the server node to be deleted. Required.

run-server

Starts an xDB server node.

Usage: xdb run-server [options]

Argument	Description
--address ARG	Listen address. The address argument can be used on a multihomed host for a ServerSocket that will only accept connect requests to one of its addresses. The value "*" means the server will accept connections on any/all local addresses.
--port ARG	Port number of the server node.
--nodename ARG	Name of the server node to start. Optional. If nodename is not specified, the command will attempt to start the primary node.
--force	If specified, ignore errors during recovery and mark offending libraries as unusable.
-f ARG --federation ARG	Path to bootstrap file.

run-server-repair

Starts an xDB server node in a special mode suitable for index repair.

Usage: xdb run-server-repair [options]

The arguments are the same as in the case of the run-server command.

stop-server

Stops a running xDB server node.

Note: This command only supports remote shutdown, and the primary node should be running when this command is executed.

Usage: xdb stop-server [options]

Argument	Description
--nodename ARG	Name of the server node to stop.

update-node

Updates an existing xDB server node.

Usage: xdb update-node [options]

You must specify at least one of the arguments logpath, port, and host, otherwise there is nothing to update.

Argument	Description
<code>--logpath ARG</code>	New log directory for the server node to be updated. Optional. If not specified, the log path will not be updated. If specified, the directory must be accessible to both the primary and the specified non-primary nodes.
<code>--port ARG</code>	New port number of the server node to be updated. Optional.
<code>--host ARG</code>	New host name of the server node to be updated. Optional.
<code>--nodename ARG</code>	Name of the server node to be updated. Required.

Library-related commands

add-binding

Binds a library to a specified xDB server node in addition to all the existing bindings.

Usage: `xdb add-binding [options] path`

Argument	Description
<code>path</code>	Path of a library which will be bound to the specified node in addition to all the existing bindings. Supports multiple libraries, separated by space.
<code>--nodename ARG</code>	The name of the node to which to bind the library (read-only libraries can bind to multiple nodes). Required.

add-file

Adds a new database file to a database segment.

Usage: `xdb add-file [options] segmentid`

Argument	Description
<code>segmentid</code>	The unique ID of the segment to which the file will be added.
<code>--path ARG</code>	The path to server side directory where the database file should be created. If not specified, the file will be created in the path for the federation default database.

Argument	Description
<code>--maxsize ARG</code>	The maximum size (in bytes) that the file is allowed to grow to. Default is 0. A value of 0 means the size is unlimited.
<code>-d ARG --database ARG</code>	Database name. Required.

add-segment

Adds a segment to the database.

Usage: `xdb add-segment [options] segmentid`

Argument	Description
<code>segmentid</code>	The unique ID of the segment to be added. Required. Supports multiple segments, separated by space.
<code>--path ARG</code>	The path to the location where the default file of the segment should be created. If not specified, then the path is the same as that for the federation default database.
<code>--maxsize ARG</code>	The maximum size (in bytes) that the file is allowed to grow to. Default is 0. A value of 0 means the size is unlimited.
<code>--temp</code>	If specified, create a new segment for temporary data.
<code>-d ARG --database ARG</code>	Database name. Required.
<code>--nodename ARG</code>	The name of the server node to which the segment should be bound. Defaults to primary node.

attach-library

Attaches a library back into a database from a detached state.

Usage: `xdb attach-library [options] segmentid target`

Argument	Description
<code>segmentid</code>	ID of the segment which contains the root page of the library. Required.
<code>target</code>	Path of target library which the attached library is appended to. Required.
<code>-d ARG --database ARG</code>	Database name. Required.

backup-library

Backs up one or more libraries within a database.

Usage: xdb backup-library [options] path

Argument	Description
path	Path(s) of library or libraries to backup. Required. Supports multiple libraries, separated by space.
--overwrite	Overwrite output file if it exists. Optional.
-o ARG --file ARG	Output file.
-d ARG --database ARG	Database name. Required.

change-binding

Binds a library to a specified xDB server node.

Usage: xdb change-binding [options] path

Argument	Description
path	Path of a library which will be bound to the specified node. Supports multiple libraries, separated by space.
--nodename ARG	The name of the node to which to bind the library. Required.

detach-library

Detaches a library if it is detachable.

Usage: xdb detach-library [options] path

Argument	Description
path	Path of a library to detach. Supports multiple libraries, separated by space.
-d ARG --database ARG	Database name. Required.

force-detach

Unconditionally detaches a segment on which the library has been created.

Usage: xdb force-detach [options] segment

Argument	Description
segment	The ID of the segment on which the library was created.
-d ARG --database ARG	Database name.

remove-binding

Removes a specified server node as one of the binding nodes of a library.

Usage: xdb remove-binding [options] path

Argument	Description
path	Path of a library one of whose binding nodes will be removed. Supports multiple libraries, separated by space.
--nodename ARG	The name of the node which will be removed as one of the binding nodes of the library. Required.

remove-segment

Deletes a segment from the database.

Usage: xdb remove-segment [options] segmentid

Argument	Description
segmentid	The ID of the segment to remove. Required. Supports multiple segments, separated by space.
-d ARG --database	Database name. Required.

restore-library

Restores one library or multiple libraries within a database from a backup.

Usage: xdb restore-library [options] path

Argument	Description
path	Path of one library or multiple libraries to restore, separated by space. Optional. If not specified, restore all libraries in the backup file.

Argument	Description
<code>--overwrite</code>	Overwrite data files if they exist. Optional.
<code>--file</code>	Input file. Optional. If not specified, use standard input.

set-file-maxsize

Sets the max size of a segment database file.

Usage: `xdb set-file-maxsize [options] path`

Argument	Description
<code>path</code>	The full path of the database file (as produced by the <code>show-segment</code> command).
<code>--segment ARG</code>	The id of the segment which the datafile belongs to.
<code>--maxsize ARG</code>	The maximum size (in bytes) that the file is allowed to grow to. A value of 0 means the size is unlimited. Required.
<code>-d ARG --database ARG</code>	Database name. Required.

set-library-state

Sets the state of a library.

Usage: `xdb set-library-state [options] path`

Argument	Description
<code>path</code>	Path of a library whose state will be changed. Supports multiple paths, separated by space.
<code>--searchable true false</code>	Set the library to searchable or non-searchable. Optional. If not specified, the library search-state remains unchanged.

Argument	Description
<code>--readonly true false [--recursive]</code>	Set the library state to read-only or read-write. Optional. If not specified, the library read-state remains unchanged. If the <code>--recursive</code> option is specified, set specified read-state on the library including the descendants; if not specified, only set state on this library.
<code>--unusable</code>	Set the library state to unusable. Optional. This option cannot be used together with options <code>--readonly</code> and/or <code>--searchable</code> .

show-segment

Shows all properties of a segment, including the segment paths. Properties will be separated by space.

Usage: `xdb show-segment [options] segmentid`

Argument	Description
<code>segmentid</code>	The ID of the segment whose properties will show. Required. Supports multiple segments, separated by space.
<code>--datafile true false</code>	If true, show only information about the database files in the segment, including full path, maximum file size and current file size.
<code>-d ARG --database ARG</code>	Database name. Required.

show-unusable-libraries

Shows full paths of all unusable libraries in specified database.

Usage: `xdb show-unusable-libraries [options]`

Argument	Description
<code>-d ARG --database ARG</code>	Database name.

General commands

ls

Lists database contents.

Usage: xdb ls [options] path

Argument	Description
path	Path to perform the operation on. Supports multiple paths, separated by space.
--type ARG	Only list contents of the given type. Must be either document, library, or detachable (only list detachable library children).
--details	List detailed properties in columns separated by tabs. If this option is specified, all library properties are shown, including segment IDs.

show-backup-metadata

Shows metadata related to a specified backup file, or to all backup files in a specified folder. Metadata fields will be separated by spaces.

Usage: xdb show-backup-metadata [options] path

Argument	Description
path	File path. If the path denotes is a normal backup file, show metadata of the backup file; if the path denotes a folder, show backup metadata of every file in the specified folder.
--recursive	Show backup metadata of every file in the specified folder and its subfolders recursively. Ignored if the path denotes a normal backup file.

show-unusable-indexes

Shows information regarding all unusable indexes in a specified database.

Usage: xdb show-unusable-indexes [options]

Argument	Description
-d ARG --database ARG	Database name.

Creating and restoring backups

It is best practice to back up your xDB federations on a regular basis, to minimize data loss in case of a system failure. The xDB administration tools provide backup/restore functionality for:

- creating and restoring full and incremental backups of federations
- backing up and restoring libraries
- (de)serializing libraries and documents

A backup made while any xDB code is running is called an online backup, or **hot** backup.

A backup made while no xDB code is running is called an offline backup, or **cold** backup.

Federation backup and restore

Possible federation backup scenarios include:

- **Online backup:** xDB backs up an entire active federation. For creating and restoring online (hot) backups, you can use any of the following xDB tools:
 - the **xdb backup** and **xdb restore** commands of the command line client
 - the backup and restore dialogs of the Admin Client
 - the Ant tasks `<backup/>` and `<restore/>`
- [Incremental backup, page 263](#): xDB backs up only the changes since the most recent online or incremental backup. **Note:** xDB provides a **standalone** option to back up the entire federation without affecting the current sequence of incremental backups.
- [Offline backup, page 266](#): provided that no xDB code is running on a federation, a conventional file backup utility can be used for a cold backup of the entire inactive federation. An alternative is to run the xdb backup command specifying the federation bootstrap file as the federation. If xDB code is running on the federation, use of external backup tools is not a good choice. Even if no transactions are open, the server can still flush dirty pages from the cache to the database files during the backup, which could result in data inconsistency in the backup files.
- [Snapshot backup, page 266](#)

To allow for a low-level software or hardware tool to take snapshots of an active federation, xDB write activity can be temporarily suspended.

By default, xDB restores all files to the backup location, and does not overwrite existing files. The following restrictions apply to restoring backups:

- Before restoring incremental backups, a full backup must be restored.
- Incremental backups must be restored in the order they were created.
- Any existing database files must be deleted or moved manually before restoring a federation.

Note: You must restore online backups using the same xDB version that they were taken on. If you want to restore a database/federation from a backup taken on an older xDB version, you must do so using that xDB version, then cleanly shutdown the server and upgrade.

Note: A federation server must never be started before the last incremental backup has been restored. It is not possible to restore any further incremental backups after starting the server.

Running incremental backups

Incremental backups store only the data that has been modified since the most recent full or incremental backup.

The federation's primary transaction log is stored in the backup file. By default, any log files that are no longer needed for transaction rollback or recovery are automatically deleted.

To allow incremental backups, the `keep-log-files` option of the federation must be enabled before creating the initial full backup. You can set this manually, either:

- in the Admin Client, by using the Set keep-log-files option dialog, accessible through the Federation menu's option Change keep-log-file option.
- in the bootstrap file, by setting the keep-log-files attribute of the log element to **true**. **Note:** The bootstrap file can only be edited if xDB is not running.

When the `keep-log-file` option is enabled, obsolete log files are only removed when a backup is created.

You can perform incremental backups with the Admin Client or the Command Line Client:

- in the Admin Client, select the Backup option of the Federation menu, and set the Incremental option of the Backup dialog,
- in the Command Line Client, use the `--incremental` option of the **xdb backup** command, [page 263](#).

Note: An incremental backup is only valid relative to the latest full backup that precedes it. If you need to do a full backup without disturbing your current sequence of incremental backups, create a *standalone backup*. A standalone backup does not affect the next incremental backup, and can be created by running the command `xdb backup` with the `--standalone` flag. It is not possible to create an incremental backup relative to a standalone backup.

Note: During incremental backup of a federation that includes a library with MultiPath index, final merge logging optimization should be disabled. You can enable or disable optimization on the federation by setting the MultiPath indexing property `xdb.lucene.finalMergeNoLogging` to true (default) or false. For more information, refer to the section about [MultiPath index merge](#), [page 156](#).

The xdb backup command

The **xdb backup** command creates an online backup of a federation in a single backup file. If this command is used on a running server, the server has to be configured for remote clients. The command must use the `xhive://host:port` format for the remote bootstrap property.

Table 63 Options for the xdb backup command

Argument	Description
<code>-o --file ARG</code>	Specifies the output file. If no output file is specified, the output is sent to standard output.
<code>--incremental</code>	Creates an incremental backup.
<code>--standalone</code>	Creates a standalone backup.
<code>--keeplogfiles</code>	Keeps obsolete log files after completing the backup.
<code>--include-segments</code>	Comma-separated list of either segments to be included or segments to be skipped during backup. Segments are specified as <code>database:segmentID</code> . The include and skip options are mutually exclusive.
<code>--skip-segments</code>	
<code>--include-segments-file</code>	Location of a file containing comma-separated list of either segments to be included or segments to be skipped during backup. Segments are specified as <code>database:segmentID</code> . The include and skip options are mutually exclusive.
<code>--skip-segments-file</code>	
<code>--overwrite</code>	Overwrites an existing output file.

Example

The following example creates an incremental backup and writes the output to the `xdb_backup.bak` file.

```
xdb backup --federation xhive://localhost:1235 --incremental --file xdb_backup.bak
```

If the amount of backup data is large, it is faster to back up from the same JVM as the page server is using. Backing up from the same JVM avoids sending all the data over a TCP connection. If such a backup on the server side is not practical, a possible alternative may be to stop the page server and run a backup directly on the federation by passing the `--federation path/to/FederationFile.bootstrap` option to the backup command.

The xdb restore command

The **xdb restore** command restores a federation backup from a single backup file. To restore a full backup and corresponding incremental backups, the command must be run for each backup file, in the order in which the backups were created. For command options refer to [table Command line options for xdb restore command, page 264](#).

Table 64 Command line options for the xdb restore command

Argument	Description
<code>--federation <value></code>	The new bootstrap file location. If no location is specified, the federation is restored to the same location from which it was backed up, including configuration values from the <code>xdb.properties</code> file. Relative paths in the original bootstrap file are interpreted relative to the new bootstrap file. Database files specified with absolute paths are restored to their original location.
<code>--file <value></code>	The name of the input file containing the backed up data. If no input file is specified, the input is read from standard input.

Argument	Description
<code>--overwrite</code>	Overwrites database files that already exist in the target federation. If not set, restore fails if any of the database files already exists, to protect data from accidental overwriting.
<code>--relative-mapper</code>	Maps restored paths relative to the restored bootstrap file.
<code>--configurable-mapper <value></code>	A file that maps new restore paths to the old paths. If no path is provided an error will be raised. For an example, see below.

Example of a configurable mapper file

```
<xhive-configurable-restore>
<restore-path>
  <old-path>
    log
  </old-path>
  <new-path>
    C:\foo\log
  </new-path>
</restore-path>
<restore-path>
  <old-path>
    MyDatabase-default-0.XhiveDatabase.DB
  </old-path>
  <new-path>
    C:\foo\MyDatabase-default-0.XhiveDatabase.DB
  </new-path>
</restore-path>
</xhive-configurable-restore>
```

Restoring lost data from log files

If the data files have been lost, the current log files are intact, and the `keep-log-files` option was set, the log files can be used to restore the database.

To restore lost data from log files:

1. Move the current log files to a safe place.
2. Restore the federation, the full backup and any available incremental backups. Do not start the server yet.
3. Check the numbers in the file names of the restored log files and the current log files. There may be overlap, but there should be no gap. If there is a gap, either not all incremental backups were restored or the `keep-log-files` option was not set at any time since the last incremental backup, and the data cannot be restored.
4. Copy the numbered log files from the current log files to the directory with the restored log files. Overwrite any numbered log files with identical names in the restored log file directory. Do **not** copy the `xhive_checkpoint.wal` file. The `xhive_id.wal` files should be identical.

5. Start the server. This should use all the log files to recover the state of the federation to the most recent one.

Related topics

[The xdb restore command](#)

Viewing backup metadata

When you create a backup, metadata about the backup is stored in a backup header. In the Admin Client, you can use the **Backup information** option of the **Federation** menu to view this backup information.

Backup metadata includes:

- The type of backup. Possible types are: FEDERATION, STANDALONE, INCREMENTAL, and LIBRARY.
- A textual description of the backup, for example:
"Library backup for database db1 created at 9/23/10 4:11 PM, backup LSN = 15609. Included libraries: /library1".
- The time when the backup was created.
- Last backup LSN (Log Sequence Number).
- If it is a federation backup:
 - Is it standalone, or incremental?
 - Which libraries are excluded?
- If it is a library backup:
 - Which libraries are included?
 - Which database?

Offline backups

If no xDB code is running, a federation can be backed up and restored using any regular file backup and restore utility. Another option is to run the `xdb backup` command using an in-JVM server by specifying the federation bootstrap file as the federation.

If xDB code is running on the federation, a cold backup is not a good choice. Even if no transactions are open, the server can still flush dirty pages from the cache to the database files during the backup, which could result in data inconsistency in the backup files.

Related topics

[Using the xdb backup command](#)

Suspending xDB activity for snapshot backups

Federation snapshots can be created using any appropriate software or hardware method. After creating the snapshot, any regular file backup utility or the xDB backup command can be used to back up the federation files.

If the snapshot is atomic, no special measures are required. The disk image of the federation files is always in a consistent state. If the snapshot is not atomic, all xDB write activity can be temporarily suspended to take a consistent snapshot of the federation. For example, if several snapshots of different file systems are required to back up a federation.

From the command line, you can use the **xdb suspend-diskwrites** command. This command supports the following options:

Option	Description
<code>--flush</code>	Flushes all dirty pages in the cache to the disk.
<code>--checkpoint</code>	Takes a lightweight checkpoint. If used together with the <code>--flush</code> option, it takes a heavyweight checkpoint. If creating a backup while disk writes are suspended, this ensures that redo recovery is not necessary after restoring the backup. The checkpoint option is ignored on replicators. Replicators cannot take independent checkpoints.
<code>--sync</code>	Flushes all files to the disk. This option is useful if you use a low level backup mechanism that bypasses the operating system when copying the federation files.
<code>--resume</code>	Resumes disk writes after suspension.

Backing up and restoring a library

Detachable read-only libraries can be backed up and restored manually using the commands **xdb backup-library** and **xdb restore-library**.

Possible uses of library backup and restore include:

- Disaster recovery, for example as protection against a library having been corrupted due to media failure.
- Archiving, for example when a library will not be used for a long time. In such cases, after the library is backed up, it is usually detached, and the library data files are removed from database. The backup file can be saved in a less expensive storage.

Note: Only the database administrator can perform library backups and restores. Any attempt to back up a non-detachable library, or a detachable library that is not read-only, causes an exception. Libraries cannot be backed up incrementally.

By default, all files will be restored to the location that they were in when the backup was made. Existing files will not be overwritten, unless the `-overwrite` option is specified. It is also possible to restore the library to a new location on the same or a different machine.

Depending on the situation before a library backup, and the required result after restore, the restored library may need to be attached, have its name or other properties changed, and/or have its consistency checked.

Commands for backing up and restoring a library

The **xdb backup-library** command creates a backup from one or more read-only detachable libraries to a single backup file or to the standard output. The library backup can be restored using the **xdb restore-library** command. For more information about these commands, refer to [Library-related commands, page 255](#).

Example

The following examples use these commands to back up and restore a library.

```
xdb backup-library --file un_charter_backup.bak "/UN Charter"  
xdb restore-library --federation xhive://localhost:1235 --file un_charter_backup.bak
```

Methods for creating and restoring backups

The xDB API backup/restore functionality discussed below includes:

- creating and restoring full and incremental backups of federations
- backing up and restoring libraries
- reading of backup metadata
- connecting to a federation backup
- serialization and deserialization of libraries and documents

Using the backup() method

The **backup()** method of **XhiveFederationIf** creates an online ("hot") backup of the federation. It can only be called if a server is running. The **xdb backup** command is a simple wrapper for the API method.

To create an incremental backup, use the backup method with the **BACKUP_INCREMENTAL** option. To allow incremental backups, the **keep-log-files** option of the federation must be enabled before creating the initial full backup, using the **setKeepLogFiles()** method of the **XhiveFederationIf** interface.

Example

The following example code uses the **backup()** method to create an online federation backup.

```
XhiveSessionIf session = XhiveDriverFactory.getDriver().createSession();
```

```

session.connect("superuser", "password", null);
XhiveFederationIf federation = session.getFederation();
FileOutputStream out = new FileOutputStream("backupfile");
federation.backup(out.getChannel(), 0);

```

Using the restoreFederation() method

The restoreFederation() method in the XhiveFederationFactoryIf interface can be used to restore a federation backup.

Example

The example code below restores a federation from a backup file.

```

XhiveFederationFactoryIf federationFactory = XhiveDriverFactory.getFederationFactory();
FileInputStream in = new FileInputStream("backupfile");
federationFactory.restoreFederation(in.getChannel(), null, null);

```

Using the library backup() method

A read-only detachable library can be backed up using the **backup()** method of the **XhiveLibraryIf** interface.

Example

The following example backs up a library.

```

XhiveSessionIf session = XhiveDriverFactory.getDriver().createSession();
session.connect("administrator", "password", "database");
DomLibrary lib = get a handle to the library for which to create the backup;
if (lib == null) {
    // throw exception
}
FileOutputStream out = new FileOutputStream("backupfile");
lib.backup(output.getChannel());

```

Using the restoreLibrary() method

A detachable library can be restored from a backup file or from standard input using the **restoreLibrary()** method.

Example

The following example restores a library.

```

XhiveFederationFactoryIf federationFactory = XhiveDriverFactory.getFederationFactory();
FileInputStream in = new FileInputStream("backupfile");

```

```
factory.restoreLibrary(input.getChannel(), "bootstrapfile", null);
```

Backing up and restoring multiple libraries

The following method in `XhiveDatabaseIf` allows you to create a backup containing multiple detachable libraries:

```
void backupLibrary(Collection<XhiveLibraryIf> libraries,
WritableByteChannel out)
```

This method backs up the specified libraries to the file specified by *out*. If *out* is null, the backup goes to the standard output. The specified libraries must all be detachable and read-only.

You can restore a backup containing multiple libraries using the normal restore process.

To selectively restore a library from a backup containing multiple detachable libraries, use the following method from `XhiveFederationFactoryIf`:

```
void restoreLibrary(ReadableByteChannel in, Collection<String>
librariesToRestore, String bootstrapFilename, PathMapper mapper)
```

This method restores the libraries specified in *librariesToRestore* (each represented by a full path string), from a backup *in*. If *librariesToRestore* is null, the method restores all the libraries in the backup. If any specified libraries are not found in the backup, the method throws an exception.

Reading backup metadata

When you create a backup, metadata about the backup is stored in a backup header. In the Admin Client, you can use the **Backup information** option of the **Federation** menu to view this backup information.

You can use the `XhiveBackupInfoIf` interface to read this backup header. To get an object of this type populated with the header information about a specified backup, use the `XhiveFederationFactoryIf.getBackupInfo` method. You can then read the backup metadata using the following methods of `XhiveBackupInfoIf`:

Table 66 API methods for reading backup metadata

Methods	Description
<code>BackupType getBackupType()</code>	Returns the backup type.
<code>String getDescription()</code>	Returns a textual description of the backup, for example: "Library backup for database db1 created at 9/23/10 4:11 PM, backup LSN = 15609. Included libraries: /library1".
<code>Date getCreationTime()</code>	Returns the time of creation.
<code>String getDatabase()</code>	Returns the database name of the libraries included in the backup. This method is applicable only to library backups.
<code>Collection<String> getExcludedLibraries()</code>	Returns a collection of full paths of detachable libraries that are excluded from the backup. This method is applicable only to federation backups. In federation backup, the library full path takes the qualified format, for example: db1:/library1.

Methods	Description
Collection<String> getIncludedLibraries()	Returns a collection of full paths of detachable libraries that are included in the backup. This method is applicable only to library backups.
long getBackupLSN()	Returns the backup LSN.

Connecting to a federation backup

The xDB API makes it possible to obtain an xDB driver object that operates directly on a federation backup, without having to restore the backup beforehand. This backup driver provides a read-only federation view on the backup data and can be used for any read-only operation such as checking the consistency of the backup or querying the backup using XQuery.

The following types of backups are supported:

- Standalone federation backup
- Full federation backup, optionally followed by one or more incremental backups

Using a backup driver is similar to using a regular driver:

1. Obtain a backup driver by using the **getBackupDriver()** method of the **com.xhive.XhiveDriverFactory** class:

```
File workDir = ...;
SeekableByteChannel full = ...;
SeekableByteChannel incremental1 = ...;
SeekableByteChannel incremental2 = ...;
XhiveDriverIf driver = XhiveDriverFactory.getBackupDriver(workDir, full, incremental1,
```

The *workDir* argument represents a working directory that will be used for storage of temporary data resulting from processing the transaction log files in the backup. If *workDir* is null, the operating system default temporary directory will be used.

The **getBackupDriver()** method takes a sequence of backups; in the code example above, a full federation backup and two incremental backups are provided.

2. Initialize and use the backup driver like a regular driver:

```
driver.init();
XhiveSessionIf session = driver.createSession();
session.connect(UserName, UserPassword, DatabaseName);
```

```
// perform read-only operations
...
```

```
driver.close();
```

```
try (SeekableByteChannel full = ...;
     SeekableByteChannel incremental1 = ...;
     SeekableByteChannel incremental2 = ...) {
    XhiveDriverIf driver = XhiveDriverFactory.getBackupDriver(null, full, incremental1, inc
    driver.init();
```

```
XhiveSessionIf session = driver.createSession();
session.connect(userName, userPassword, databaseName);
session.begin();

// perform read-only operations
...

session.commit();

driver.close();
}
```

API documentation

[com.xhive.XhiveDriverFactory](#)

[com.xhive.core.interfaces.XhiveDriverIf](#)

[com.xhive.core.interfaces.XhiveSessionIf](#)

Serialization and deserialization of libraries and documents

Serialization and deserialization can be done using the Admin Client or the following API calls:

- **XhiveLibraryChildIf.serialize(OutputStream out)** - Serializes a library child to an output stream.
- **XhiveLibraryIf.deserialize(InputStream in)** - Deserializes a library child, changing it to a child of the calling library.
- **XhiveDatabaseIf.deserializeRootLibrary(InputStream in)** - Deserializes a library, changing it to the root library of the calling database.

For more information about these methods, see the API documentation.

Managing detachable libraries

A library can be detached from the database by calling the **XhiveLibraryIf.detach()** method. When a library is detached from a database, the library is logically removed from the database and becomes non-accessible until it is attached. Once detached, a library may be physically removed from the database or moved into a different storage.

Creating a detachable library requires calling the **XhiveLibraryIf.createLibrary(int options, String segmentId)** method with the **XhiveLibraryIf.DETACHABLE_LIBRARY** option flag. When this flag is set, a segment ID must also be specified. Moreover, the segment must already exist, and cannot have been used previously by any other library, even if that previous library has been deleted. Once used for a detachable library, a segment can be used for that library only. A used segment cannot be shared with other detachable libraries.

There are several basic methods for managing detachable libraries, as described in the table [API methods for managing detachable libraries, page 272](#).

Table 67 API methods for managing detachable libraries in XhiveLibraryIf

Methods	Description
isDetachable()	Identifies whether a library is detachable.

Methods	Description
<code>detach(String)</code>	Detaches a library if it is detachable. The library is logically removed from the database once it is detached.
<code>attach(String)</code>	Attaches a detached library to the original database from which it was detached. When a detached library is attached to the original database, it can be attached to the original location or moved to a different location.
<code>forceDetachChild(String)</code>	Forcibly detaches a child library. This method is designed to be used only when the child library (and/or any descendants of this child library) is believed to be corrupted.
<code>forceAttach(String)</code>	Attaches a library which is previously detached through forceDetachChild(String) or has one or more currently unusable segments back into the database from detached state.
<code>getState()</code>	Identifies the state of a library.
<code>setState(LibraryState)</code>	Changes the state of a library. Changing the state of a detachable library to read-only also changes the state all library children to read-only. Changing the state of a detachable library to read-write causes an exception if any of its ancestors is in a read-only state.
<code>getAllSegmentIds()</code>	Gets the ids of all the segments occupied by a detachable library.
<code>getBindingNodes(boolean)</code>	Gets the names of all the binding nodes of a detachable library.
<code>addBinding(String)</code>	Binds a read-only detachable library and all its descendant libraries to the specified node in addition to all its existing bindings.
<code>remove(String)</code>	Removes the specified node as one of the binding nodes of a read-only detachable library and all its descendant libraries.
<code>changeBinding(String)</code>	Binds a detachable library and all its descendant libraries to the specified node.
<code>backup(WritableByteChannel)</code>	Creates a backup of a read-only detachable library.
<code>getNonSearchable()</code>	Identifies whether a library is searchable.
<code>setNonSearchable(boolean)</code>	Changes a searchable library to a non-searchable library.

Sample

[DetachLibrary.java](#)

Related links

[Using the library backup\(\) method](#)

[Using the restoreLibrary\(\) method](#)

API documentation

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

Moving a detachable library

A detachable library can be moved from one database to another provided both databases belong to the same federation. This is accomplished by first detaching the library from its current database and then attaching it to its new database.

To move a detachable library programmatically, use the regular **XhiveLibraryIf.detach()** call to do the detach step first (note that this is done in a session connected to the source database):

```
lib.setState(LibraryState.READ_ONLY, true);
lib.detach();
```

Once the library is successfully detached, use the special **XhiveLibraryIf.attach(String, String, String, XhiveFederationFactoryIf.SegmentIdMapper)** call in another session which is connected to the destination database to actually move the library:

```
session2.connect(administratorName2, administratorPassword2, databaseName2);

// begin the 2nd database admin transaction
session2.begin();
XhiveLibraryIf root2 = session2.getDatabase().getRoot();

// create a simple segment id mapper (optional)
XhiveFederationFactoryIf.SegmentIdMapper mapper =
    new XhiveFederationFactoryIf.SegmentIdMapper() {
        @Override
        public String getSegmentId(String originalSegmentId) {
            return originalSegmentId + "_2";
        }
    };

// move the library from the 1st database to the 2nd database
root2.appendChild(root2.attach(databaseName1,
                                administratorPassword1, segmentId, mapper));

session2.commit();
```

This special attach API is different from the regular **XhiveLibraryIf.attach(String)** API as it needs to access the data of a different database. As a result, the caller at the same time has to be the administrator of the source database and provides the administrator credential in the API call.

The segments of the library are moved into the destination database together with the library. The ID of a segment is unique only within the database in which it was originally created, so there may be an ID collision when moving a segment to another database. Should it be necessary, an **XhiveFederationFactoryIf.SegmentIdMapper** object can be specified to rename the IDs of the segments in the destination database.

The data files of the segments are also moved, logically. Since xDB adopts a naming convention which combines database name, segment ID and file ID for all its data files, the moved data files will violate this convention after the move. xDB will attempt to rename them under certain conditions or the next time server restarts.

Sample

[MoveLibrary.java](#)

API documentation

[com.xhive.dom.interfaces.XhiveLibraryIf](#)

[com.xhive.core.interfaces.XhiveFederationFactoryIf.SegmentIdMapper](#)

Unusable detachable libraries

If a detachable library is found to be corrupted for any reason (such as a media failure), the library and its descendant libraries can be marked as *unusable* by setting a Boolean *usable* property to false. Any library marked as unusable will be skipped during query and search processing. The *usable* property is applicable to detachable libraries only; an attempt to set the property on a non-detachable library will cause an exception. Further, a detachable library can be set to unusable only if it is a child of a concurrent library.

You can check whether a library is corrupt by performing a consistency check on it, and then set the library to unusable or usable accordingly. The database administrator should run consistency checker before creating a backup as well as after a backup has been restored.

You can detect a corrupted library at runtime by catching an `XhiveDataCorruptionException` (which extends `XhiveException`).

There are several methods for managing detachable library usability, as described in the table below:

Table 68 API methods for managing detachable library usability in XhiveDatabaseIf

Method	Description
<code>setLibraryUsableBySegmentId(String segId, boolean usable)</code>	Mark the detachable library whose root page is on the specified segment as usable if <i>usable</i> is true, unusable otherwise.
<code>setLibraryUsableByPageId(long pageId, boolean usable)</code>	Mark the detachable library that contains the specified page as usable if <i>usable</i> is true, unusable otherwise.
<code>setLibraryUsableByPath(String fullPath, boolean usable)</code>	Mark a detachable library, which is specified by the full path, as usable if <i>usable</i> is true, unusable otherwise.
<code>getAllUnusableLibraries()</code>	Return a collection of strings representing the full paths of all unusable libraries in the specified database.
<code>getAllUnusableExternalIndexes()</code>	Return all unusable <code>MultiPath</code> indexes in the database.

In addition, the `XhiveFederationFactoryIf` interface exposes the following method:

`getAllUnusableLibraries(String bootstrapFile, String database)`

which returns a collection of strings representing the full paths of all unusable libraries in the specified database.

An unusable library will be skipped when a query is executed against the library or any of its ancestors. The `library.getChildren()` method will exclude any unusable children in its result.

Duplicated transaction log files

To provide additional protection for the transaction logs, xDB can be set up to automatically maintain one or more duplicates of a server node's transaction log files in separate locations. The original log location is called the primary log directory, and the duplicates are called secondary log directories. Each

server node always has one single primary log directory, and one or more secondary log directories can be added. Secondary log directories can be removed; the primary log directory cannot be removed.

For performance reasons, it is preferable to have each transaction log location on a separate disk. However, even if all duplicates are on the same disk, increased redundancy can still help protect the transaction logs against the consequences of I/O errors, file corruption, and so on.

Note: When keeping duplicates of transaction log files, xDB concurrently writes the same information to multiple identical log files. Duplicating transaction log files therefore increases the amount of I/O that the page server must perform. Depending on your configuration, this may impact overall performance.

To recover from a failure on the primary log directory, first shut down the page server, then copy the duplicate log files from a secondary log directory to the primary log directory, and then restart the page server.

xDB writes to the available secondary log directories while ignoring any unusable ones. If xDB cannot write to a secondary log directory, it marks that log directory as "unusable" in the bootstrap file and places an error message in the message log. Writing to the primary log directory and any other secondary log directories proceeds normally.

Log directories can be added or removed using command line client [commands, page 248](#).

Methods for transaction log duplication

You can create a federation with multiple log directories, and/or add a log directory to an existing node, as follows:

1. Create an `XhiveLogConfigurationIf` object:

```
XhiveFederationFactoryIf ff = XhiveDriverFactory.getFederationFactory();
XhiveLogConfigurationIf logConfig =
    ff.createLogConfiguration(PRIMARY_LOG_DIR, SECONDARY_LOG_DIRS_LIST);
```

2. Use the object to create a federation.

```
ff.createFederationWithLogConfig(BOOTSTRAP, logConfig,
    PAGE_SIZE, SUPERPWD);
```

3. Or, to add a new node:

```
XhiveSessionIf session = driver.createSession();
session.connect(SUPERUSER, SUPERPWD, null);
session.begin();
XhiveFederationIf federation = session.getFederation();
federation.addNodeWithLogConfig(nodeName,
```

```

    hostName, portNum, logConfig);
session.commit();

```

Modify the log directories configuration of an existing node:

1. Retrieve the XhiveLogConfigurationIf object of the node:

```

XhiveSessionIf session = driver.createSession();
session.connect(SUPERUSER, SUPERPWD, null);
session.begin();
XhiveFederationIf federation = session.getFederation();
XhiveLogConfigurationIf logConfig =
    federation.getNodeLogConfiguration(NODE_NAME);

```

2. Add (or remove) a log directory:

```

logConfig.addLogDirectory("sLog1");

```

Monitoring statistics

xDB implements extensive monitoring capabilities, which can be used to analyze various aspects of xDB performance.

xDB monitors statistics separately for each server, which allows for better understanding of each server's load.

xDB supports monitoring of various [statistics categories, page 279](#), including number of cache hits, page access, query response time, number of connected transactions and transaction rollback time. Monitoring can be enabled and disabled for various categories, either on a system or on transaction level.

By default, monitoring is disabled. It can be enabled or disabled through the property `XHIVE_STATISTICS_MONITORING_ENABLED` in the `xdb.properties` file, which is set to `false` by default.

If you set this property to `true` and then run an xDB server, then monitoring will be enabled for that server.

With statistics monitoring enabled, you can use the Command Line Client command **monitor-statistics** with the following arguments:

- **path:** The connection URL to the local JMX server (required).
- **password:** The password to the local JMX server (required).
- **username:** The username to the local JMX server (required).
- **duration:** The amount of time (in seconds) we monitor, infinite if empty.
- **interval:** Statistic data polling intervals (in seconds). Defaults to `XHIVE_STATS_MONITOR_INTERVAL`.

- **nodename:** The server node to monitor (required).
- **federation:** The federation to monitor (required).
- **category:** The category to monitor. If left empty, all categories are monitored. Available category options are: xdb, server, transaction, session, replicator, query, bufferpool, page.

Enabling statistics monitoring

By default, statistics monitoring is disabled. You can enable/disable monitoring either through:

- **a Superuser/Administrator session** If the session is connected as an Administrator/Superuser, it can enable the categories using the `XhiveSessionIf.enableStatistics(String category, boolean enable)` call. Categories enabled/disabled by this session will be so for the entire server. The category provided can also be a prefix of several other categories and this call will enable/disable all of them.
- **a user session** If the session is connected as a regular user (not an Administrator/Superuser), it can enable the categories using the same method call as above. However, it can only enable transaction level statistics, which will only be monitored for the current transaction.

Consuming monitored xDB statistics

xDB monitored statistics can be consumed in several ways (please make sure that the statistics you wish to consume are enabled on the relevant driver):

- **Through the API:** You can consume the monitored statistics of an `XhiveDriverM` by using the following API call `XhiveDriverIf.getStatisticsSubscription(String... categories)` which returns an instance of `XhiveStatisticsSubscriptionIf` which corresponds to the provided categories. You can use this instance to call `XhiveStatisticsSubscriptionIf.getSnapshot()` which returns an instance of `XhiveStatisticsSnapshotIf` which represents the values of the monitored statistics at the time of the snapshot. Each `XhiveStatisticsSnapshotIf` has two getters: `getTimeStamp()` which tell you when the snapshot was taken and `getData()` which represents a map of the values of the monitored statistics at the time they were taken. Each key represents the name of the monitored category and each value represents an `XhiveStatisticValueIf` which can be different things depending on its type.

The example code below shows how to subscribe to and receive monitored buffer pool cache statistics:

```
XhiveStatisticsSubscriptionIf subscriber =
    getDriver().getStatisticsSubscription("xdb.bufferpool");
XhiveStatisticsSnapshotIf snapshot = subscriber.getSnapshot();
Map<String, XhiveStatisticValueIf> data = snapshot.getData();
for(String category : data.keySet()) {
    System.out.println(snapshot.getTimeStamp() + ": " + category + "=" +
        data.get(category).toString());
}
```

- **Through JMX:** Each driver publishes an instance of `XhiveStatisticsMonitoringMBean` to the default platform JMX server. You can connect to this server via a JMX client and invoke the bean's `snapshot(String[] categories)` method. It will return an `XhiveStatisticsSnapshotIf` which corresponds to the `XhiveDriver` that published that bean.

Monitored statistics categories

Note: Monitored statistics categories are subject to change.

Currently, the following categories can be monitored:

- **Server:**
 - **xdb.system.receivers:** The number of receivers in a server.
 - **xdb.system.replicaids:** The replica IDs of the server.
- **Session:**
 - **xdb.system.checkpoints:** The number of checkpoints performed.
 - **xdb.system.commits:** The number of commits performed.
 - **xdb.system.rollback:** The number of rollbacks performed.
 - **xdb.system.sessions:** The number of connected sessions.
 - **xdb.system.transactions:** The number of active transactions.
 - **xdb.system.blockedtransaction:** The number of blocked transactions.
 - **xdb.system.deadlocks:** The number of deadlocks.
- **Transaction:**
 - **xdb.transaction.user:** The connected user of the transaction.
 - **xdb.transaction.database:** The database the transaction is connected to.
 - **xdb.transaction.address:** The address the transaction is connected to.
 - **xdb.transaction.node:** The node the transaction is connected to.
 - **xdb.transaction.state:** The state of the transaction.
 - **xdb.transaction.readonly:** Whether the transaction is read only.
 - **xdb.transaction.system:** Whether the transaction is a system transaction.
 - **xdb.transaction.start:** The time when the transaction started.
 - **xdb.transaction.end:** The time when the transaction ended.
 - **xdb.transaction.id:** The ID of the transaction.
 - **xdb.transaction.joinedthread:** The name of thread joined to the transaction.
 - **xdb.transaction.rollbacktime:** The time it took to rollback the transaction.
 - **xdb.transaction.committime:** The time it took to commit the transaction.
- **Replicator:**
 - **xdb.replicator.system.lastreceivedlsn:** The last LSN received by the replica.
 - **xdb.replicator.system.lastredonelsn:** The last LSN redone by the replica.
 - **xdb.replicator.system.masteraddress:** The replica master's address.
- **XQuery:**
 - **xdb.xquery.transaction.debug.optimizer:** Represent the 'optimizer-debug' output.

- **xdb.xquery.transaction.debug.queryplan:** Represent the transactional 'queryplan-debug' output.
- **xdb.xquery.transaction.debug.index:** Represent the 'index-debug' output.
- **xdb.xquery.transaction.debug.pathexpression:** Represent the 'pathexpr-debug' output.
- **xdb.xquery.transaction.debug.query:** The query text.
- **xdb.xquery.transaction.indexesused:** The indexes used in this query.
- **xdb.xquery.transaction.start:** When the query started (when execute is called).
- **xdb.xquery.transaction.timetofirstresult:** The time it took to return the first result to this query.
- **Buffer Pool:**
 - **xdb.bufferpool.transaction.hits:** The number of cache hits per transaction.
 - **xdb.bufferpool.transaction.misses:** The number of cache misses per transaction.
 - **xdb.bufferpool.system.hits:** The number of overall cache hits.
 - **xdb.bufferpool.system.misses:** The number of overall cache misses.
 - **xdb.bufferpool.system.size:** The size of the cache.
- **Page:**
 - For each page type, you can monitor the following statistics:
 - ◆ **xdb.page.system.{page type}.access**The number of overall accesses per page type.
 - ◆ **xdb.page.system.{page type}.read**The number of overall bytes read per page type.
 - ◆ **xdb.page.system.{page type}.written**The number of overall bytes written per page type.
 - ◆ **xdb.page.system.{page type}.readtime**The overall time spent reading per page type.
 - ◆ **xdb.page.system.{page type}.writetime**The overall time spent writing per page type.
 - ◆ **xdb.page.transaction.{page type}.access**The number of accesses per page type per transaction.
 - ◆ **xdb.page.transaction.{page type}.read**The number of bytes read per page type per transaction.
 - ◆ **xdb.page.transaction.{page type}.readtime**The time spent reading per page type per transaction.
 - The page types are:
 - ◆ **superblock**
 - ◆ **other**
 - ◆ **temporary**
 - ◆ **objectowner**
 - ◆ **objects**
 - ◆ **indexleaf**
 - ◆ **indexinternal**
 - ◆ **bigstring**
 - ◆ **blobleaf**
 - ◆ **blobinternal**
 - ◆ **concurrent.root**

- ◆ **concurrent.internal**
- ◆ **concurrent.leaf**
- ◆ **namebase.root**
- ◆ **namebase.internal**
- ◆ **namebase.leaf**

They and their combined prefixes are accepted.

RAM segments

A RAM segment is a special type of database segment that is kept in the database cache but never written to a file.

Note: The size of the database cache limits the amount of data that can be kept in a RAM segment. Furthermore, if too large a part of the cache is used to store temporary data, database performance can suffer due to lack of cache pages for other usages

In the Admin Client, a RAM segment for temporary data can be enabled in the Database properties dialog, by selecting the option `ram_segment` from the **Temporary data segment** dropdown list.

Read-only federations

If your xDB application requires that data remains unchanged, you can put that data in a read-only federation. For example, this can be useful for distribution on readonly media like CDROM. To facilitate such usage, xDB has some special features for readonly federations.

A federation can be changed into a read-only federation by changing the bootstrap file to a read-only file.

A read-only federation has the following characteristics:

- It requires no log files. For example, copying a federation to a CD-ROM only requires copying the bootstrap file and database files. The log files are not required. A federation can only be copied when the xDB server is not running and has been shut down cleanly. A clean shutdown allows the server to write all modified pages back to disk and the log files are not needed on startup.
- Its bootstrap file is not locked. Multiple xDB applications can be run using the same read-only federation.
- Its data cannot be modified.

Note: A read-only federation, though the data itself cannot be modified, can still allow for creation of temporary data by using a RAM segment.

Federation sets

A federation set provides a convenient way to run multiple federations in a single server with a single TCP port and a single page cache.

A federation set is defined by a *federation set description file*, which is simple XML file that contains a list of references to federations. Federation sets can be nested: a federation set description can reference other federation set description files. The example below contains absolute references to two federations and one relative reference to a (nested) federation set description file. Any relative paths to federations or nested federation sets are interpreted relative to the federation set description file.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:federation-set xmlns:ns2="http://www.xhive.com/federationset/schema">
  <federation-list>
    <federation file="D:/xDB10/data1/XhiveDatabase.bootstrap" name="UN1"/>
    <federation file="C:/xDB10/data2/XhiveDatabase.bootstrap" name="UN2"/>
  </federation-list>
  <nested-federation-set-list>
    <nested-federation-set file="../XhiveDatabase.fds" name="nest1"/>
  </nested-federation-set-list>
</ns2:federation-set>
```

A federation referenced in a federation set description file does not have to exist. Adding a federation to a federation set and creating that federation are separate and unrelated actions.

Creating a federation set

There are several ways to create a federation set description file and add federations:

- Using the `xdb federation-set` command, for example:

```
xdb federation-set create filename
xdb federation-set addfd filename --name
                             fdname --value /path/to/bootstrapfile
```

- Using the administration client and clicking **Federation sets** > **Create federation set** and entering the file name for the new federation set description file. After creating the federation set description file, the **Modify federation set** option can be used to add federations.

Using federation sets

A federation set can run on a separate server or in the application JVM. The latter option is more efficient. A federation set server can be started by running the `xdb run-server` command with the `--federationset` option instead of the `--federation` option.

The following example runs a separate federation server.

```
xdb run-server --federationset /path/to/federationset --port 1235 --cache 4096
```

Using Secure Sockets Layer (SSL)

For maximum security, xDB can run over an SSL connection. The Secure Sockets Layer (SSL) protocol can be used by web servers and web browsers for encrypted communications. SSL uses cryptographic processes to provide secure communication over a network.

Note: Encryption can have a severe performance impact.

Using SSL requires setting up a keystore and a truststore using JSSE administration and the **keytool** command.

Configuring the server for SSL

Configuring the server for SSL includes:

- Creating an SSL server socket by running the **xdb run-server** command with the **-ssl** option. Using the **-clientauth** option enables client authentication.
- Specifying system properties for the default keystore using the **-Djavax.net.ssl.keyStore=/path/to/keystore** Java command line parameter.

Configuring the client for SSL

Configuring SSL on the client includes specifying a URL as the *xhive.bootstrap* property. The URL has the format *xhives://host:port*. The system uses the default SSLSocketFactory to connect to the xDB server. It can require setting the JSSE system properties.

Checking database consistency

It can be useful to perform consistency checks, either before and/or after major events (for example before a full backup and after a restore), or on a regular basis.

Note: Use consistency checking only while there are no active updating transactions, otherwise the result of the consistency check is not defined.

The Admin Client and the command line client offer options to check the consistency of a database, federation, library, document or BLOB.

Table 69 Consistency checkers

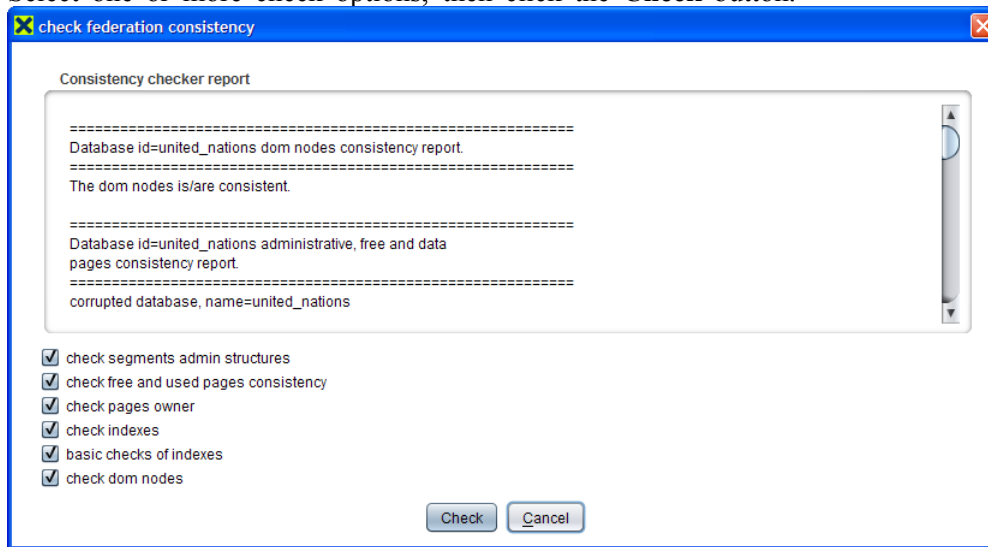
Admin Client options	CLC commands
Main menu: Federation > Check federation consistency	check-federation
Database right-click menu: Check database consistency	check-database

Admin Client options	CLC commands
Library right-click menu: Library mangement > Check library consistency	check-librarychild
XML document/BLOB right-click menu: Check document consistency Check blob consistency	check-node

In addition to checking consistency of a “live” federation, the command-line client commands check-federation, check-database, check-library-child, and check-node support also checking consistency of federation backups.

Example federation consistency check result in the Admin Client

Select one or more check options, then click the **Check** button.



When the check is complete, you can select text in the Consistency checker report and copy the selection to the clipboard.

Methods for consistency checking

The **XhiveConsistencyCheckerIf/XhiveFederationConsistencyCheckerIf** APIs provide consistency checkers, respectively for a library, database or a federation.

Note: The consistency checker can only be used when there are no active updating transactions, otherwise the result of the consistency check is not defined.

To check database consistency using the API:

1. Start a session and connect to the database as administrator.

When calling the `connect()` method, the `databaseName` parameter value is **null**.

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver();
if (!driver.isInitialized()) {
    driver.init(1024);
}
XhiveSessionIf session = driver.createSession();
session.connect(AdminName, AdminPassword, databaseName);

XhiveDatabaseIf database = session.getDatabase();
```

2. Get a consistency checker interface.

```
XhiveConsistencyCheckerIf checker = database.getConsistencyChecker();
```

3. Create a `PrintWriter` object that contains the consistency checker report, similar to the following:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
PrintWriter pw = new PrintWriter(baos);
checker.setPrintWriter(pw);
```

4. Check and report database consistency, similar to the following:

```
if (checker.checkDatabaseConsistency()) {
    System.out.println("Database: " + database.getName() + " is consistent");
} else {
    pw.flush();
    System.out.println(baos.toString());
}
```

Checking federation consistency

Given an active session, the code sample below shows how to check a federation.

```
XhiveFederationIf fed = session.getFederation();
XhiveFederationConsistencyCheckerIf checker = fed.getConsistencyChecker();
ByteArrayOutputStream baos = new ByteArrayOutputStream();
PrintWriter pw = new PrintWriter(baos);
checker.setPrintWriter(pw);
if (checker.checkFederationConsistency().isConsistent()) {
    System.out.println("Federation at driver: " +
        session.getDriver().getFederationBootFileName() + " is consistent");
} else {
    pw.flush();
    System.out.println(baos.toString());
}
```

Message logging

Message logging provides run-time status information in a human-readable form, and is unrelated to [write ahead logging, page 43](#). This section applies to the **java.util.logging** logging back-end, which is the default xDB message logging implementation. For more information about Java's logging system, refer to the Javadoc documentation for the class **java.util.logging.LogManager**, or the documentation of your JVM. **Note:** To facilitate adaptation to environments that depend on specific logging frameworks or configurations, xDB uses SLF4J as message logging framework. For more information about SLF4J, refer to [Message logging framework, page 287](#). If you want to use xDB with a different logging implementation, refer to that implementation's documentation.

The **java.util.logging** functionality relies on the global logging configuration for the JVM in which it runs. By default, log messages with a priority of INFO or higher are written to the console. Java's logging system is typically configured through configuration files, either a default one per JVM in `JAVA_HOME/lib/logging.properties`, or a file specified with the system property `java.util.logging.config.file`, for example on the Java command line. For the xDB standalone server, you can configure this in the additional Java VM option `XHIVE_OPTS` in `xdb.properties`, or in the lax configuration file (see [Configuration files for Windows, page 68](#)).

For example, you can enable logging for the `xdb` command, by adding `-Djava.util.logging.config.file=[logging.properties]` to `JAVA_CMD`.

Under Windows in `xdb.bat`:

```
"!JAVA_CMD!" -Xms!XHIVE_MIN_MEMORY! -Xmx!XHIVE_MAX_MEMORY! !XHIVE_OPTS! \
-Djava.util.logging.properties.file=mylogging.properties \
-cp "!XHIVE_CLASSPATH!" com.xhive.tools.Cmd %*
```

Under UNIX in `xdb.sh`:

```
"${JAVA_CMD}" -Xms${XHIVE_MIN_MEMORY:-128M} -Xmx${XHIVE_MAX_MEMORY:-256M} \
${XHIVE_OPTS} \
-Djava.util.logging.properties.file=mylogging.properties \
-cp "${CLASSPATH}" com.xhive.tools.Cmd "${@}"
```

You can configure multiple handlers to receive logging information (for example, to write log messages to a file and to send them by email). You can set log levels for individual logging areas, identified by a hierarchical package name. Logging areas are hierarchical package names, separated by `?.?`. Areas cascade: setting a log level on `com.xhive` will apply to all sub-packages, unless they have their own log level specified.

The example below configures a console handler, and specifies different logging levels for several logging areas.

```
# specify a set of handlers, in this case just a console handler
handlers = java.util.logging.ConsoleHandler
# log level for a particular handler
java.util.logging.ConsoleHandler.level = INFO
# log level for xDB core messages
com.xhive.core.level = FINE
# less information from multi path indexes
com.xhive.index.multipath.level = SEVERE
```

Message logging areas

xDB's message logging areas include:

- **com.xhive.core** - messages from the database kernel, typically about critical failures of core database processes. Do not set the log level above SEVERE for this logger.
- **com.xhive.index.multipath.indexing** - messages concerning the administration and indexing process of multi path indexes.
- **com.xhive.index.multipath.query** - messages about queries against multi path indexes.
- **com.xhive.index.multipath.merging** - messages about merging activities in multi path indexes.
- **com.xhive.trace.rpc** - trace messages about RPC operation in client/server mode, typically on level FINEST. For more information, refer to [RPC tracing, page 78](#). **Note:** Enabling these can have an adverse performance impact.

Message logging framework

xDB uses SLF4J (see www.slf4j.org) as message logging framework (with the exception of the xDB Ant tasks, which use Ant's built-in logging system).

SLF4J is a facade for Java logging frameworks such as java.util.logging, Log4J, logback and commons-logging. SLF4J decouples the logging API from the actual logging implementation, making it possible to plug in a different logging implementation at deployment time, without having to modify the application code. This makes it easy to adapt xDB to environments that depend on specific logging frameworks or configurations.

The logging in SLF4J is configured through the underlying logging implementation. How this is done depends on the specific logging framework.

The xDB distribution uses **java.util.logging** as the default logging implementation. To use a different logging framework, remove the lib/xhivedb/core/slf4j-jdk14.jar SLF4J binding library from the Java classpath and substitute it with the desired SLF4J binding library.

Replicating Federations

This chapter contains the following topics:

- **Replication**
- **Creating a federation replica**
- **Running a replicator on a dedicated server**
- **Replication of federation metadata**
- **Changing a replica into a master**
- **Removing a replica**
- **Methods for replication**

Replication

Load sharing by spreading database reads over multiple page servers can provide a large performance advantage. xDB supports this by means of *lazy primary copy replication*.

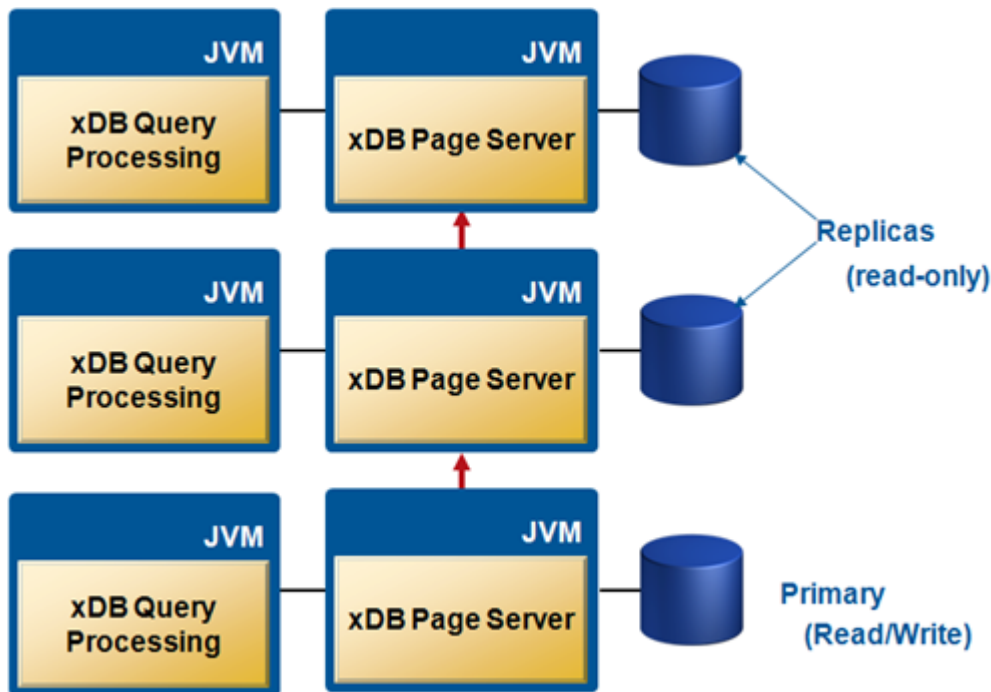
With xDB, *replication* involves maintaining one or more full copies of a federation, with each copy having a separate page server. One page server maintains a “master” copy of the data, while one or more additional page servers each maintain a read-only “slave” copy of the data.

Primary copy replication means that data updates occur on a master federation, known as the primary or the primary copy. Applications write data only to the primary. The updates propagate from the master to one or more read-only copies of the primary, known as replicas, secondary copies, or slaves. One primary can have multiple replicas. For each replica, there is a separate, dedicated page server called a replicator or replication server. A replicator can act as master for another replica. Applications can perform read-only transactions and online backups on the replicas instead of on the primary, to distribute query load and improve performance. Also, applications can provide for using a replica as a failover.

The term *lazy* indicates that the master does not wait for a transaction to propagate to the replicas before confirming the transaction. The replicas are updated asynchronously in the background.

The replication process sends the master’s primary transaction log files to the replicators. The replicators read the log files and apply all updates to their own copy of the data. The master preserves any required log files until the replica has confirmed that it received the files.

Primary with two replicas



Creating a federation replica

To create a replica from a federation, its replicator must be given a name that is registered with the master. This tells the master that a replica will be added, and will make the master preserve any required transaction log files until that replica confirms that it received the log files.

To register a replicator manually, you can use the Admin Client.

After registering the replicator, you can create a copy of the federation in one of the following ways:

- Use the **xdb create-replica** command from the primary federation. If desired, the replicated federation can then be moved to another machine. Alternatively, you can run the command on the destination machine, connecting to the original server remotely, like this:

```
xdb create-replica --federation xhive://masterhost:1235
--replicaid newOrExistingReplicaId
--replicabootstrappath /path/to/new/replicadata/XhiveDatabase.bootstrap
--password superuserpassword
```

The example above assumes that the master server is running a page server that is accessible from other hosts, and that the replica host has xDB installed.

- Use **Replicate federation** option of the Admin Client.
- Use the **xdb backup** command or the Admin Client to create a full online backup, and restore that backup on the desired machine.

- If no server is running for the federation, you can copy all the federation files to another machine using a system file copy command.

Note: Do not make a system file copy while a server is running for the original federation, because that will corrupt the data in the replica.

First registering a replicator name with the master and then creating the copy of the federation implies this replicator name is also registered with the copy. The replicator server preserves its log records for another replicator with this name, unless the name is unregistered from the copy.

To unregister a replicator name with a copy, use the Admin Client or the **xdb create-replica** command with `--remove` option.

Running a replicator on a dedicated server

To run a dedicated page server as a replicator, duplicating all updates from the master, use the **xdb run-server** command with the following options to provide the master location and the registered name of the replicator:

- `--master xhive://host:port`

This option passes the location of the master page server. The connection uses the same port as regular client connections. Specifying an https protocol identifier creates an SSL connection to the master.

- `--replicator replicatorId`

This option passes the replicator name. The replicator name must be registered with the master.

Clients can connect to the replicator server for read-only queries and online backups.

Replication of federation metadata

Federation metadata is stored in the federation bootstrap file. Not all this data is replicated, because it can be necessary to use different settings at the different copies of the federation.

The following federation metadata is replicated:

- Superuser password, license key

This type of metadata can only be updated at the master and is replicated to the secondary copies automatically.

- Databases, segments, and files

By default, the replica uses the same path to the file as the original. Relative paths are always interpreted relative to the bootstrap file. If different paths are required for the secondary copies, it is usually easiest to use symbolic links those paths to point to the desired actual directories, provided your system supports symbolic links. Otherwise, a path mapper can be used. For more information, see the API documentation.

The following federation data is **not** replicated:

- Updates to the `keep-log-files` option

This option can be set independently at the master and the replicas. This method is useful if only one of them is used to create incremental backups. In that case, the update option is set to **true** only for the copy that is used for incremental backups.

- Registering and unregistering of replicators

This type of metadata can be stored in any copy and is not replicated. For example, registering replicator X at replicator Y causes replicator Y to retain the log records. The records are retained until the registered replicator X has confirmed that the master has received the logs.

Changing a replica into a master

Moving the role of master copy to one of the replicas is useful if the master server is taken out of service.

To turn a replica into a master, the master must be synchronized with the replica by shutting down the master copy and specifying the replica as new master.

The master copy can be shut down manually using the Admin Client or the **xdb stop-server** command. By default, the master is synchronised with all registered replicators. If the `--wait` option is used, the master is synchronised with the listed replicators only. After the old and new master have been synchronized, a regular master server can be run on the new master.

Example

The example below uses the `xdb stop-server` command to shut down a server, after updating its two replicas.

```
xdb stop-server --federation xhive://dbserver:1235 --wait "id1, id2"
superuserpwd
```

Passing an empty string as `--wait` option instructs the process not to wait for any replicators:

```
xdb stop-server --federation xhive://dbserver:1235 --wait ""
superuserpwd
```

Removing a replica

A replica is removed by stopping the replication server and removing the files.

Subsequently, the replicator must be unregistered from the master, so the master does not preserve the log records for this replica. Otherwise the master preserves obsolete log files forever. A replicator can be unregistered manually using the Admin Client or the **xdb create-replica** command with `-remove` option.

Methods for replication

Methods for using a federation replica

To create a replica from a federation, its replicator must be given a name that is registered with the master to inform the master that a replica is added. The **registerReplicator** method of the **XhiveFederationIf** interface registers a replicator.

After registering the replicator, you can create a copy of the federation using one of the following methods:

- Create a copy of an online federation directly, using the **XhiveFederationIf.replicateFully** API.
- Create a full online backup, using the **XhiveFederationIf.backup** API, and restore the backup on the machine that will host the replicator.

Use the **XhiveFederationIf.unregisterReplicator** method to unregister a replicator.

If the **XhiveFederationIf.shutdown** method is used to shut down a master copy, it will wait until all specified replicators are completely up to date with this server. If necessary, it will also wait for replicators to connect.

Running a replicator on an internal server

If a replicator is used for queries, it is more efficient to use an internal server. An internal replicator server is started using the API. Internal replicator sessions are created like other sessions, except that internal replicator sessions can only perform read-only transactions.

Example

The following example starts an internal replicator server.

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver("/xhive/replica/
XhiveDatabase.bootstrap");
driver.configureReplicator("xhive://masterhost:1234", "myReplicator");
driver.init(1024);
```

Read-only transactions with temporary data

Read-only transactions can still create temporary data, such as new nodes in XQuery queries or old versions of versioned documents. The replication mechanism updates all data in normal segments, so the replica cannot allocate temporary data to normal segments.

Therefore, to use temporary data in the replica, you must create a temporary data segment in the replica and use that segment for its temporary data. Updates for temporary data at the master are not logged, and therefore not replicated, which means that the replicator can safely use its own temporary data segment for its own temporary data.

Note: Instead of a file, you can allocate a RAM segment for temporary data by using the **setTemporaryDataSegment** method with the special value **RAM_SEGMENT_NAME**. However, a RAM segment is useful only if there are enough cache pages to hold the temporary data of all simultaneous sessions, with some cache space to spare for other usages.

Using a replica as a failover

If the master copy fails for any reason, one of the replicas can take over the role of the master. xDB does not provide tools to detect failure, because an xDB tool is useless for an application server with an internal xDB server. Failure detection has to include the entire application and initiate the failover process.

If there is only one replica, it can become the master by stopping the replicator and starting a dedicated or internal xDB server. The following example describes using the API.

```
driver.close();
driver.configureReplicator(null, null);
driver.init(cachePages);
```

The new master automatically performs any necessary log record updates that were already transferred and rolls back any updates of uncommitted transactions.

For xDB 7.3 versions and later, it is also possible to turn a replicator into a master server without stopping the server. Running read-only transactions can continue normally. Use the administration client or the API, as described in the following example.

```
XhiveSessionIf session = driver.createSession("make master");
session.connect("superuser", password, null);
XhiveFederationIf federation = session.getFederation();
federation.turnReplicatorIntoMaster();
```

If there are any other replicators, they must be stopped and restarted with the URL of the new master server. The replicators must have been registered with the new master before the failure.

Once the old failed master is up running, it can take over the role of the master. This method requires a full replication back from the new master and configure the old master as a replica, even if the data of the old master was preserved. The old master could contain updates that were not yet replicated at the moment of failure. These updates can interfere with the updates on the new master and corrupt the data.

Preparing for replication

This example shows how to set up a simple replicating system based on an existing single server application. In this example, the specific goal of replication is to improve performance by duplicating the application and federation on different hosts.

First prepare the databases for replication by creating a temporary segment in each database. That way the temporary results can be created in the replica databases.

1. To create a temporary segment, use code like the following:

```
// Creating a temporary segment in Java code
session.getDatabase().createTemporaryDataSegment("tempSegmentName", null, 0);
session.getDatabase().setTemporaryDataSegment("tempSegmentName");
```

A temporary segment can also be created with the Admin Client: just right-click **Segments**, enter an ID and check the **temporary** flag.

2. Create the replica by running the following command on the replica host:

```
xdb create-replica --federation xhive://masterhost:1235
--replicaid newOrExistingReplicaId
--replicabootstrappath /path/to/new/replicadata/XhiveDatabase.bootstrap
--password superuserpassword
```

The example assumes that the master server is running a page server that is accessible from other hosts, and that the replica host has xDB installed. The `create-replica` command is a wrapper around the **XhiveFederationIf.registerReplicator(...)** and **XhiveFederationIf.replicateFully(...)** API calls. The calls register the replica in the master federation and copy the complete federation over to the replica host.

Replication application code sample

On the master host, the federation can be accessed in the normal way, as long it acts as a page server to the replicating host. Assuming for performance reasons that the database files are accessed directly, the driver initialization code in the application is:

```
XhiveDriverIf driver =
    XhiveDriverFactory.getDriver("/path/to/XhiveDatabase.bootstrap");
driver.init(numCachePages);
driver.startListenerThread(new ServerSocket(1235));
```

It is best to have one code base for the application. Use a configuration switch to indicate whether the machine is acting as a replica or master, with a corresponding driver configuration and usage code.

The driver configuration code on the replica host depends on the application. If the application only runs read-only transactions on the replica federation, then only an extra call to the **XhiveDriverIf.configureReplicator(...)** object is needed. However, in this example the application also allows for read-write transactions. Since xDB replication only permits updating of the master copy of the federation, this process requires configuring the replica to allow for update transactions. Specifically, on the replica a distinction is made between read-only transactions and update transactions, which have to access the master federation. This results in much slower performance for these update transactions. Therefore this setup is only advised if most of the application transactions are read-only transactions.

The following code creates two driver objects, one that acts on the replica and one that connects to the master-server directly:

```
// On replica-host, use two drivers
XhiveDriverIf masterDriver =
    XhiveDriverFactory.getDriver("xhive://masterhost:1235");
masterDriver.init(numCachePages / 2);
XhiveDriverIf replicaDriver =
    XhiveDriverFactory.getDriver("/path/to/replicadir/XhiveDatabase.bootstrap");
driver.configureReplicator("xhive://masterhost:1235", "previouslySetupId");
replicaDriver.init(numCachePages / 2);
```

In the session pooling code, you decide which driver to get the session from based on whether it is a read-only transaction or not. For performance, it is advisable to have your session pool code and the code that uses the session pool already distinguish between read-only and read-write transactions. Use the **XhiveSessionIf.setReadOnlyMode(...)** method for concurrency performance. The following example describes the `getSession`.

```
synchronized XhiveSessionIf getSession(boolean readOnlyTransaction) {
    if (readOnlyTransaction) {
```

```
        return waitForUpdates(replicaDriver.createSession());
    } else {
        return masterDriver.createSession();
    }
}
```

The code that pools the sessions replaces the `createSession` call. Since there are two drivers, also set up two session collections.

The read-only transactions are faster because they use the replica. However, xDB replication is lazy. When an update transaction on the master federation finishes, it is not guaranteed that a transaction started directly after on the replica detects the changes. This restriction can be problematic, if the application adds a document to a library and then immediately presents the contents of the library to the user in a read-only transaction. The library contents could not be updated on the replica yet. Therefore it is better for a session on the replica to wait for updates made in a session on the master server. In the session pool code, you could register a timestamp each time a read-write session completes:

```
XhiveSessionIf.Timestamp currentWaitTimeStamp = null;

synchronized returnSession(XhiveSessionIf session) {
    // Regular session pool code
    ....
    if (session.getDriver().equals(masterDriver)) {
        currentWaitTimeStamp = session.getUpdateTimeStamp();
    }
}
```

Then call a routine from the `getSession` method that waits for the updates to be available on the replica before continuing:

```
XhiveSessionIf waitForUpdates(XhiveSessionIf session) {
    if (currentWaitTimeStamp != null) {
        session.waitForTimeStamp(currentWaitTimeStamp);
        currentWaitTimeStamp = null;
    }
    return session;
}
```

Note: **Lazy** does not mean that replication changes take a long time. It means that replication is not guaranteed. In practice, the `XhiveSessionIf.waitForTimeStamp(...)` call completes immediately or quickly.

Configuring Multiple Backend Servers

This chapter contains the following topics:

- **xDB multi-node architecture**
- **Transaction recovery**
- **Multi-node bootstrap configuration**
- **Multi-node considerations**
- **Multi-node run-time restrictions**
- **Managing nodes**
- **Methods for multi-node deployment**
- **Replacing a non-primary node**
- **Changing node identity**
- **Replacing a primary node**

xDB multi-node architecture

The multi-node feature can provide greater flexibility when it comes to creating a highly scalable XML database deployment, with higher availability and disaster recovery capabilities. Multi-node can also provide great benefits for ingesting large volumes of data.

This chapter introduces multi-node concepts and constraints, gives some examples of possible multi-node deployment and use, and discusses some disaster recovery aspects.

A page server is responsible for delivering data pages, which store documents and indexes, to front-end applications. By default, xDB allows only a single page server per federation.

In some cases, a single page server can become a bottleneck, for example due to an increase in the number of clients. The multi-node feature allows you to distribute a data set over multiple so-called *node servers*, in a way that is somewhat analogous to partitioning in the relational database world. Its data partitioning mechanism is based on [Detachable libraries, page 46](#). A data binding mechanism is used to assign each detachable library to a node server. A node server can service multiple libraries. A library can be bound to several node servers. When a library is bound to a node server, that node server can read and modify the library.

The data is distributed into a number of different detachable libraries, each serviced by a particular node server. Any reading or writing of the detachable library data pages must go through the binding node server. A calling application, such as a web application hosted within an application server, can invoke xDB to store and retrieve data in the libraries in much the same way as with a single page server, and xDB transparently dispatches requests to the relevant nodes.

If data is evenly distributed among the various libraries, you can achieve parallel ingestion of the data.

One of the node servers is designated as *primary server*, and the other node servers are called non-primary servers. Usually, each node server will run on a separate host machine, but multiple node servers can run on a single host if necessary.

Note: The primary server must always be active to enable access to the database.

Although data to node server binding is implemented at library level, the actual binding information is stored at segment level. Changing the binding of a library causes updates to all the segments of the library.

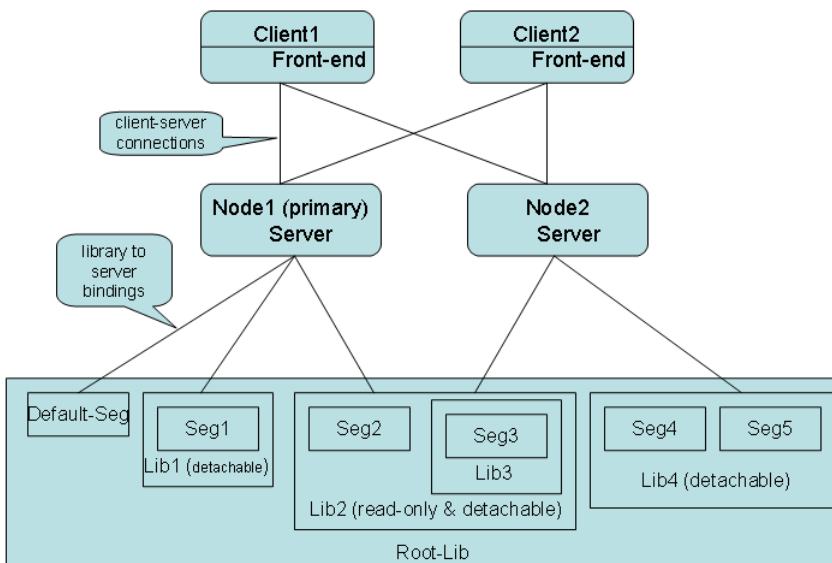
Example of a multi-node configuration

An example configuration with two xDB nodes is shown in [xDB multi-node configuration, page 298](#).

This example shows two front-end application servers, each hosting a web application with an xDB client within it. Behind the scenes, each xDB client can connect to both of the server-side nodes. Each node is bound to one or more libraries. One of the node servers is designated as the primary server, the other node server is called a non-primary server.

Node server 1 serves the detachable libraries lib1, lib2, and lib3. As primary, it also serves the root library, which cannot be detachable. Node server 2 serves detachable libraries lib2, lib3, and lib4.

Figure 10 xDB multi-node configuration



Transaction recovery

Each node server maintains its own transaction log and recovers the libraries that are bound to it. A transaction in a multi-node installation can only make updates on a single node. All transactions are handled the same way as in a single-node installation. If a node server fails, restarting the node server recovers the binding libraries to a consistent state.

Note: The primary node server must always be started first.

Multi-node bootstrap configuration

A multi-node configuration requires a modified bootstrap file format, using `<node/>` and `<binding_server/>` elements to provide information about the nodes and the segment-to-server binding.

Note: The command-line client and the Admin Client provide functions to modify the bootstrap file for multi-node. **Editing the bootstrap file manually is NOT recommended.**

Multi-node bootstrap file example

The sample bootstrap file below defines the example configuration shown in [xDB multi-node configuration, page 298](#).

The node and segment-to-server bindings are highlighted. There are two xDB nodes. The `name="primary"` attribute for the first node indicates that this is the primary server. The federation has six segments. The default segment and Seg1 are bound to the primary. Segments Seg4 and Seg5 are bound to Node2. The two read-only segments Seg2 and Seg3 are bound to both nodes.

```
<?xml version="1.0" encoding="UTF-8"?>
<server version="xDB 10.1" pagesize="8192" license=<license> passwd=<password>>
<node name="primary">
  <log path="log" id="1210583277531" keep-log-files="false"/>
</node>
<node name="Node2" host="host2", port="1236",>
  <log path="Node2-log" id="1210583888574" keep-log-files="false"/>
</node>
<database name="Shanghai">
  <segment id="default" temp="false" version="1" state="read-write"
    usage="non-detachable" usable="true">
    <file path="Shanghai-default-0.XhiveDatabase.DB" id="0"/>
    <binding_server name="primary"/>
  </segment>
  <segment id="Seg1" temp="false" version="1" state="read-write"
    usage="detachable_root" usable="true"
    library-path="/Lib1" library-id="0">
    <file path="Shanghai-Seg1-0.XhiveDatabase.DB" id="1"/>
    <binding_server name="primary"/>
  </segment>
  <segment id="Seg2" temp="false" version="1" state="read-only"
    usage="detachable_root" usable="true"
    library-path="/Lib1/Lib2" library-id="0">
    <file path="Shanghai-Seg2-0.XhiveDatabase.DB" id="2"/>
```

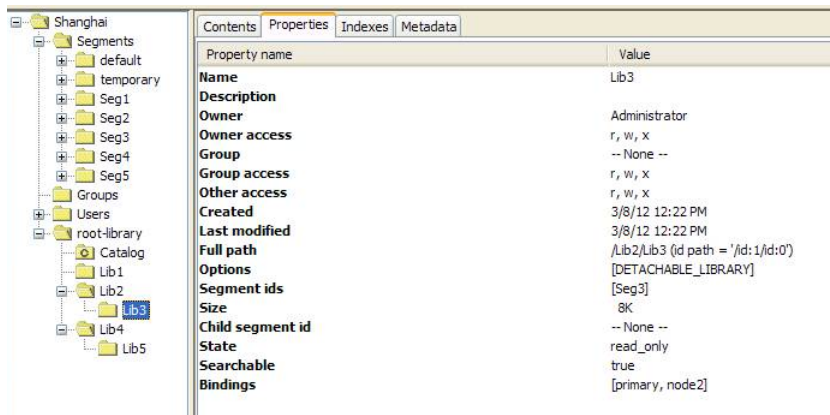
```

    <binding_server name="primary"/>
    <binding_server name="Node2"/>
</segment>
<segment id="Seg3" temp="false" version="1" state="read-only"
  usage="detachable" usable="true"
  library-path="/Lib1/Lib2" library-id="0">
  <file path="Shanghai-Seg3-0.XhiveDatabase.DB" id="3"/>
  <binding_server name="primary"/>
  <binding_server name="Node2"/>
</segment>
<segment id="Seg4" temp="false" version="1" state="read-write"
  usage="detachable_root" usable="true"
  library-path="/Lib1/Lib4" library-id="1">
  <file path="Shanghai-Seg4-0.XhiveDatabase.DB" id="4"/>
  <binding_server name="Node2"/>
</segment>
<segment id="Seg5" temp="false" version="1" state="detach_point"
  usage="detachable_root" usable="true">
  <file path="Shanghai-Seg5-0.XhiveDatabase.DB" id="5"/>
  <binding_server name="Node2"/>
</segment>
</database>
</server>

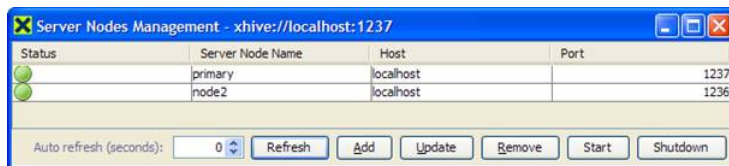
```

Multi-node Admin Client example

To the Administrator, the above bootstrap example would look like this in the Admin Client:



The **Server nodes** option of the **Federation** menu would show:



Server upgrade from older xDB release

When an xDB 10 server is started for the first time against a federation that was created using an older xDB release, an internal server upgrade process modifies the bootstrap file. By default, the upgraded server acts as the primary server. All existing libraries are not detachable, and are bound to the upgraded server.

If the resulting federation is to be changed into a multi-node configuration, the administrator has to set up each node manually.

<node/>

The <node/> element describes a node server.

Attributes

Name	Default	Description
node name		The name of the node server. The node element for the primary node does not have the host and port attributes. The node name of the primary node is always primary . The name is automatically assigned to the primary node when a federation is created and cannot be changed.
host name		The host name of the node server, if the server is not the primary node.
port number		The port number of the node server, if the server is not the primary node.
node server log directory		The log directory of the node server. The default log directory of primary server is log . The default log directory for a non-primary node server is <i>host name-log</i> . With multi-node support, the "log" element is a child of the "node" (instead of the "server") element. When a node server is added, the administrator can specify a different log directory for that node server.

Child elements

The <node/> element can have the following child elements:

- <log/>

<binding_server/>

The <binding_server/> element maps a segment (library) to multiple node servers.

Segment to server bindings are stored as child elements of <segment/>, [page 231](#) element in the bootstrap file.

Attributes

Name	Default	Description
name		The name of the node server to which the segment is bound.

Multi-node considerations

For multi-node, the recommended indexing method is multipath indexing.

Multi-node constraints

Multi-node architecture imposes a number of constraints, including:

- Multi-node configuration requires use of [client-server mode, page 69](#).
- Multi-node configuration does not support replication.
- For access to the database, the primary node server must always be active.
- Data storage must be symmetrically accessible to all the server hosts. This implies that all the database files, including server logs, must be stored on SAN, NAS, or NFS.
- All node servers must use the same directory path to access the data directory.
- Only characters a-z, A-Z and 0-9 are allowed in server node names.
- The following multi-node library binding rules are enforced:
 - A library is always bound to at least one node server.
 - The root library of a database is always bound to the primary node server and the binding cannot be changed.
 - A read-write detachable library can be bound to only one node server.
 - A read-only detachable library can be bound to more than one node server.
 - Binding a detachable library to a node server binds all descendant libraries, including non-detachable libraries, to the same server.
 - Only the binding of a detachable library can be changed. The binding of a non-detachable library cannot be changed directly. However, if the non-detachable library has a detachable ancestor, the binding can be changed indirectly by modifying the binding of the detachable ancestor.
 - Only the database administrator can change the bindings of a detachable library.
- Multi-node does not fully support distributed transactions. An update transaction can make updates only to libraries bound to a single node server, but it can read data pages in any libraries of the database provided it follows locking rules to avoid distributed deadlock.

Example

The table below compares two examples of valid transactions with one invalid transaction, occurring on the libraries in the [multi-node architecture, page 297](#) example.

Transaction 1	Transaction 2	Transaction 3
begin	begin	begin
...
update Root-Lib	read Root-Lib	update Lib1
update Lib1	read Lib2	update Lib4
...	read Lib3	...
commit	...	commit
	commit	
Transaction 1 is valid, because the Root library and Lib1 are bound to the same server.	Transaction 2 is also valid, because it performs no updates.	Transaction 3 is not allowed. It violates the locking rules, because it updates Lib1 and Lib4, which are bound to different nodes.

Multi-node run-time restrictions

The following run-time restrictions apply to the primary and non-primary node servers:

- The primary server must always be started first. Non-primary servers can be started in any order.
- The primary server must be up and running to access the federation.
- When the primary server is down, the entire federation becomes unavailable.
- When a non-primary server is down, the libraries that are bound to the server become unavailable.
- A node server can be started only after the node server has been added.
- A node server can be updated or removed only if the node server is not running.

Managing nodes

xDB automatically creates a primary node as part of a federation. The primary node cannot be removed.

The command-line client provides **xdb** commands to manage and configure multi-node architecture properly, including **add-node**, **remove-node**, and **update-node**, **add-binding**, **change-binding** and **remove-binding**. The commands **run-server** and **stop-server** have multi-node options.

Note: If you need to modify the bootstrap file for multi-node, use xDB functions. Editing the bootstrap file manually is NOT recommended.

Command examples

The following examples show use of the **xdb add-node**, **xdb remove-node**, and **xdb update-node** commands to add, remove, and update node servers.

```
xdb add-node --passwd secret --nodename "Node1" \  
-host "Node1Host" --port 1236  
xdb remove-node --passwd secret --nodename "Node2"  
xdb update-node --passwd secret --nodename "Node1" \  
-host "Node1Host" --port 1238
```

Methods for multi-node deployment

Methods for managing nodes

Non-primary nodes can be added or removed by calling the `addNode()` or `removeNode()` API methods of `XhiveFederationIf`. The `updateNode()` method updates an existing node. Non-primary servers can be shut down using the `shutdown()` method. The corresponding `xdb` commands call these API methods.

Examples

The following code fragment adds, removes, and updates a node server.

```
XhiveDriverIf driver =  
    XhiveDriverFactory.getDriver("xhive://primaryHost:1235");  
if (!driver.isInitialized()) driver.init(1024);  
XhiveSessionIf session = driver.createSession();  
Session.connect(superUserName, superUserPassword, null);  
XhiveFederationIf federation = session.getFederation();  
// Add node Node1 with host Node1Host and port 1235  
federation.addNode("Node1", "Node1Host", 1236);  
// Remove node Node2  
federation.removeNode("Node2");  
// Change listening port of Node1 to 1238  
federation.updateNode("Node1", "node1Host", 1238);
```

The following code fragment uses the `getAllNodeInfo()` method to retrieve all non-primary node server information.

```
XhiveFederationIf federation = session.getFederation();  
List<XhiveNodeServerInfoIf> nodeInfoSet = federation.getAllNodeServerInfo();  
for (XhiveNodeServerInfoIf nodeInfo : nodeInfoSet) {  
    System.out.println("Node name = " + nodeInfo.getNodeName());  
    System.out.println("Host name = " + nodeInfo.getHost());  
    System.out.println("Port number = " + nodeInfo.getPort());  
    System.out.println("Log directory = " + nodeInfo.getLogPath());  
}
```


Locking rules

In an xDB multi-node configuration, a transaction can update data pages bound to a single node only. Therefore a transaction cannot acquire write locks on more than one node. Despite this restriction, distributed deadlock is still possible. For example, a transaction can read any data pages in the database and thus acquire read locks on segments bound to any node server. Furthermore, when a transaction updates a library, it has to acquire read locks on other segments/libraries on other nodes.

Distributed deadlock can be avoided by enforcing the following locking rules:

- A transaction that has only read locks can get a read lock on any objects bound to any node server.
- A transaction can only acquire a write lock on a node server if the transaction has no write locks on the other node servers. All read locks on the other nodes must be on ancestor segments of the segment on which the transaction is requesting write lock.
- Once a transaction has a write lock on a node server, it cannot request write locks on any node servers. The transaction can request read locks only in segments that are descendants of the segment on which it has write locks, or on ancestor segments.

Disabling distributed deadlock detection

xDB can enforce [locking rules, page 305](#) to avoid distributed deadlock. Distributed deadlock detection is enabled by default.

To disable distributed deadlock detection, do one of the following:

- Set property `com.xhive.skipdistributeddeadlockdetection` to **true**.
The `com.xhive.skipdistributeddeadlockdetection` property affects all sessions in the same JVM.
- Call `setSkipDistributedDeadlockDetection(true)` of the `XhiveSessionIf` interface.
The `setSkipDistributedDeadlockDetection(true)` method call affects the calling session only

Modifying library bindings

When xDB creates a detachable library, the binding of the segment that is used determines the initial library binding. The node server to which the segment is bound can be specified when creating a segment using the `createSegment()` API method of `XhiveDatabaseIf`.

Once a detachable library has been created, you can use the `changeBinding()`, `addBinding()`, and `removeBinding()` API methods to change, add, and remove the bindings of the library.

Examples

The following code example creates a segment using the `createSegment()` method.

```
session.connect(dbaUserName, dbaUserPassword, "MyDatabase");
session.getDatabase().createSegment("node1", "segment1", null, 0);
```

The following code example changes the binding of a library to the node **Node1**.

```
XhiveLibraryIf library = session.getDatabase.getByPath("/library1");
library.changeBinding("Node1");
```

Applications for multiple-node configurations

In a multi-node configuration, each server serves only a portion of the database. Applications can get access to all the objects in the database by establishing a single session to the primary server. Applications cannot make connections to non-primary node servers directly. When the `XhiveDriver.createSession()` method creates a session, it creates a connection to the primary server. This connection is called a primary connection.

The first time a session attempts to access data on a non-primary server, it creates a connection to the non-primary server using the same user credentials. A session can have one connection to each backend server in the configuration. A connection to a non-primary server is called a sub connection. Access to different node servers is transparent to the application. This node transparency means that there are no special requirements for writing application programs intended to run in multi-node environments. As far as the application is concerned, the primary server is the only server.

An application makes connections to the primary server just like it would in a single node configuration. Every transaction on the primary server is repeated on every node server that is involved in the current transaction.

The following administrative operations should run in a separate transaction instead of executing them with other operations in the same transaction:

- Creating a detachable library.
- Changing the library state to read-only.
- The following operations in the `XhiveLibraryIf` interface:
 - `changeBinding()`
 - `addBinding()`
 - `removeBinding()`
 - `backup()`
 - `attach()`
 - `detach()`
- The `backupLibrary()` operation in the `XhiveDatabaseIf` interface.

Example application: Phone Call logging

As an example of a possible multi-node application, consider how a fictional U.S. cell phone company might use a multiple-node configuration to store cell phone call information. Each cell phone call is logged as an XML document, and all call transactions are stored in a multi-node deployment, because of the high volume of incoming data (3,000 call transactions per minute), as well as for the sake of high availability and disaster recovery.

The application logic is implemented as a distributed web application, hosted within a Java application server. The web application serves as the interface point for the company back-office systems. A back-office system will generate the call transaction and send a transaction write request to the web application. Each new call transaction will be saved as an XML document in a library for the region where the customer was when placing or receiving the call. For example, if someone in New York

City receives a call, the resulting call transaction is logged to the USNortheast library. The web application will also be the interface point for a web portal, where customers can search their personal call transaction logs. When a user enters search criteria from the portal, the portal will generate a query request to the web application.

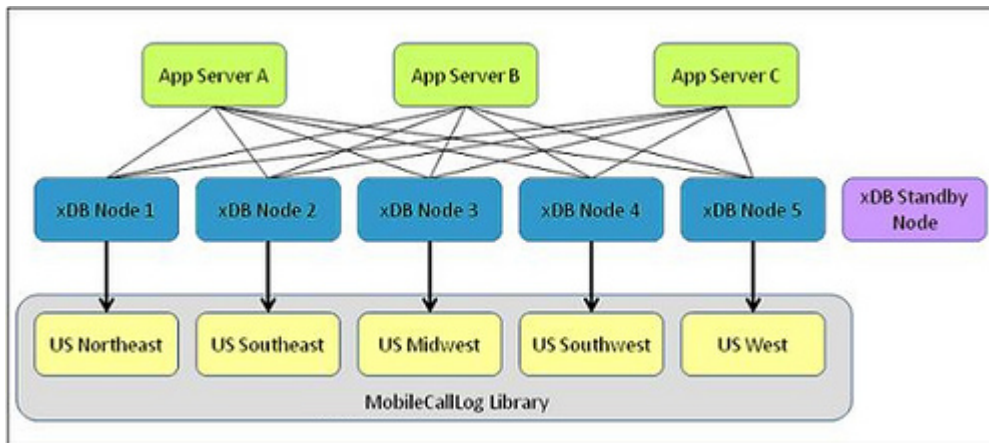
Call volumes are spread equally across the various US regions, and the company models the data as one XML library per US region, with all regional XML libraries sharing the same parent library:

MobileCallLog

- USNortheast
- USSoutheast
- USMidwest
- USSouthwest
- USWest

Deployment Topology

Due to its requirements and data model, the company chooses a multi-node deployment with 5 nodes, where each node has read/write access to a single child library:



In this deployment topology, one server node has been mapped to each regional XML library to handle read/write operations for that particular library. When the web application needs access to call transaction data in a regional library, an App Server transparently obtains a connection to the appropriate server node.

A spare server node is kept on hot standby, ready to be incorporated into the system if one of the current server nodes should fail. As a result, 6 server nodes are deployed in all. Furthermore, in this sample deployment topology the libraries reside on a SAN, therefore high availability and disaster recovery of the actual data is abstracted away from the IT owner managing the deployment.

Three identical application server instances allow requests to be load-balanced across the application servers. Each xDB client connects to appropriate server nodes as and when access to the library bound to a particular server node is required.

Data access

To explore the relationship between calling application, client APIs, and server nodes, we will walk through the system interaction for the following use cases:

- Calling application sends a request to store a call transaction to the database
- Calling application issues a query to find call transactions
- Calling application requests a specific call transaction

Store a call transaction

1. A customer in New York City finishes a cell phone call.
2. A proprietary system of the cell phone company sends a call transaction write request to one of the 3 application servers that handle call transaction write requests.
3. The application server receives the write request, and in turn invokes a client API to write the call transaction to the appropriate XML library. Because this call occurred in New York City, the calling application will have specified in the write request that the XML document should be stored in the USNortheast library. The client API transparently sends the request to Node 1, which handles read/write requests for the USNortheast library.
4. The Node 1 server receives the write request, writes the call transaction to the library, and returns success.

Issue a query

1. A customer uses a self-service web application to view his call log. In particular, the customer would like to search for call transactions within the past week.
2. Customer-facing portal dispatches the query request to one of the 3 application servers. In addition to handling call transaction write requests, the web application hosted on the 3 application servers is also designed to handle query requests.
3. The application server receives the query request and turns the query request in to a formal XQuery. The web application within the application server invokes the XQuery client API.
4. The client API transparently dispatches page requests to all 5 server nodes. In this use case, the customer's query spans across all 5 libraries because the customer did not enter a qualifier in his query such as "calls within the past week placed in NYC".
5. The client API transparently gathers pages from the 5 server nodes, continues to process the XQuery request as needed, then consolidates the results into a single XQuery result set.

6. The results ultimately get propagated from the web application up to the customer-facing portal

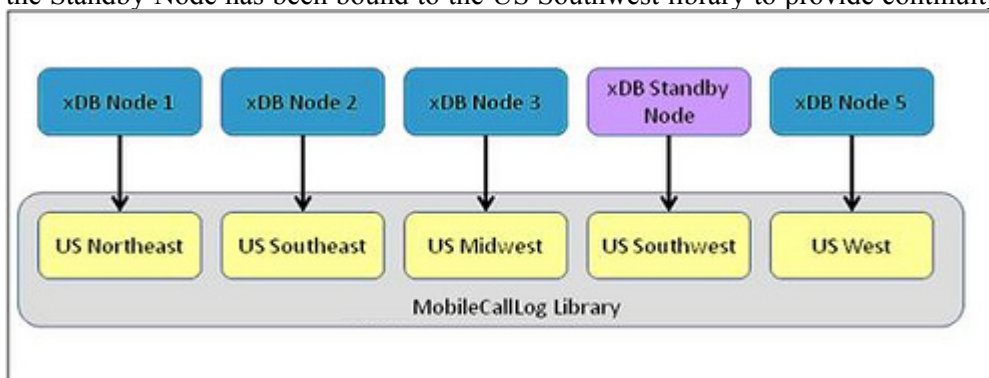
Request a specific call transaction

1. Customer uses a portal to view his call log, and would like more information about a particular call transaction. Therefore, the self-service portal issues a request to fetch a specific call transaction to one of the 3 application servers.
2. The web application within an application server invokes a client API by providing information about the call transaction, including the XML library where the underlying call transaction is called.
3. Based on the XML library specified, the client API routes the fetch request to the correct server node.
4. The server node receives the request and returns the call transaction.
5. The call transaction is ultimately propagated back up to the customer-facing portal.

Disaster recovery options

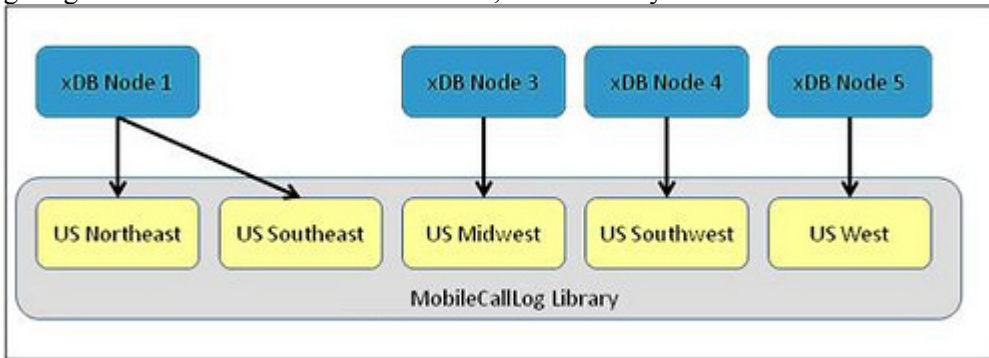
One main benefit of multi-node lies in the area of high availability and disaster recovery.

The most elaborate approach is that the multi-node deployment includes a server node whose sole purpose is to act as a hot standby: this standby is running, but not bound to any of the XML libraries. If one of the server nodes fails, the standby server node can be bound to the XML library to which the failed server was previously bound. In the figure below, Node 4 has failed, and the Standby Node has been bound to the US Southwest library to provide continuity of service.



High availability can also be achieved without a hot standby node. In the event of a node failure, another bound node can take over read access for the affected library. In the figure below, Node 2 has failed, and the US Southeast library has been bound to Node 1,

giving Node 1 access to 2 XML libraries, so continuity of service has been maintained.

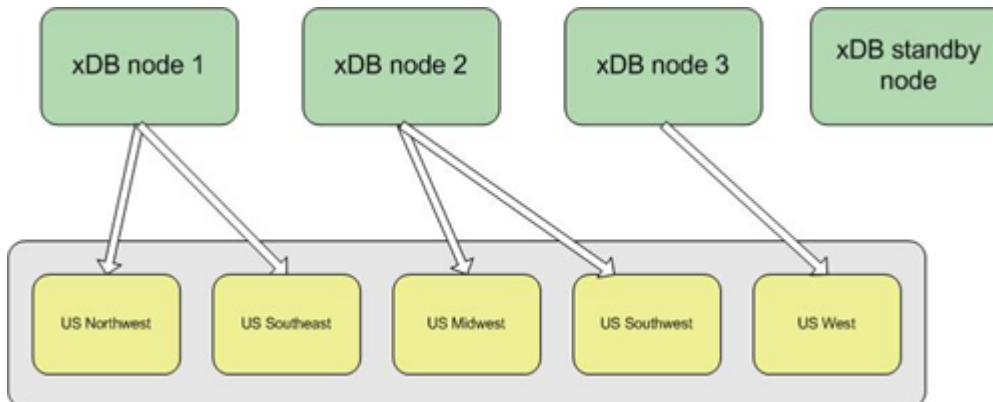


Note: These high availability techniques can only be used for read-only libraries. If a node fails that serves a read/write library, recourse is to either restart the failed node, or to replace the failed node with another preconfigured one.

An additional, worst-case disaster recovery policy should be to back up the XML libraries periodically. The XML library backup can occur in parallel. In case of a catastrophic failure, for example if the XML libraries are lost along with all the server nodes, a new environment can then be created by restoring the backup XML libraries in parallel, and bringing new server nodes online.

Multiple-node API examples

For developing and testing its new system, the phone company chooses a slightly simplified setup:



The partitioning of the documents is based on US regions, with a separate library for each region plus one hot standby node, but with only three active node servers, and some libraries sharing a node server.

Creating and starting server nodes

The fragment of java code below creates and starts the server nodes for the above multi-node architecture.

```

class MobileCallLogBackend {

    private static final String BOOTSTRAP_FILE_NAME = "some_path/fed.bootstrap";
    private static final String FEDERATION_FOLDER = "federation";
    private static final String PRIMARY_LOG_FOLDER = "log";
    private static final int PAGE_SIZE = 512;
    private static final String SUPERPWD = "test";

    private static final String XDB_NODE_PRIMARY = "primary";
    private static final String XDB_NODE_2 = "xDBNode2";
    private static final String XDB_NODE_3 = "xDBNode3";
    private static final String XDB_STANDBY_NODE = "xDBStandby";

    private static final int NODE_PRIMARY_PORT = 1235;
    private static final int NODE_2_PORT = 1236;
    private static final int NODE_3_PORT = 1237;
    private static final int NODE_STANDBY_PORT = 1238;

    /*
     * This method creates the Mobile application deployment topology.
     * It creates federation and database and then
     * creates and runs xDB nodes to serve incoming query/update queries.
     */
    private void setUp() {
        //create federation
        XhiveFederationFactoryIf ff = XhiveDriverFactory.getFederationFactory();
        ff.createFederation(BOOTSTRAP_FILE_NAME, PRIMARY_LOG_FOLDER, PAGE_SIZE, SUPERPWD);

        //start primary node
        primaryDriver = startNode(XDB_NODE_PRIMARY, NODE_PRIMARY_PORT);

        //create superuser session to create database and add new nodes
        //only superuser can create database or add/remove new nodes
        XhiveSessionIf superSession = driver.createSession();

        //connect as a superuser
        superSession.connect(SUPERUSER, SUPERPWD, null);

        //start transaction
        superSession.begin();
        XhiveFederationIf federation = s.getFederation();

        //get and change license key
        String license = getLicenseKey();
        federation.setLicenseKey(license);

        //create database
        federation.createDatabase(DBNAME, DBPWD);

        //create xDB node2, xDB node3 and standby node
        federation.addNode(XDB_NODE_2, host, NODE_2_PORT, null);
        federation.addNode(XDB_NODE_3, host, NODE_3_PORT, null);
    }
}

```

```

federation.addNode(XDB_NODE_STANDBY, host, NODE_STANDBY_PORT, null);

//commit super session
superSession.commit();

//terminate super session
superSession.disconnect();
superSession.terminate();

// start xDBnode2, xDB node3 and standby node
driverNode2 = startNode(XDB_NODE_2, NODE_2_PORT);
driverNode3 = startNode(XDB_NODE_3, NODE_3_PORT);
driverStandby = startNode(XDB_NODE_STANDBY, NODE_STANDBY_PORT);
}

/**
 * This method starts a node.
 * @nodeName name of the node to run.
 * @port specifies a port to listen to incoming requests.
 * @return driver object of the running xDB node
 */
XhiveDriverIf startNode(String nodeName, int port) throws IOException {
    //create and initialize node server
    XhiveDriverIf nodeDriver =
        XhiveDriverFactory.getDriver(BOOTSTRAP_FILE_NAME, nodeName);
    if (!nodeDriver.isInitialized()) {
        nodeDriver.init(1024);
    }

    //start listening thread on the server node to listen to incoming client requests
    ServerSocket socket = getServerSocketFactory().createServerSocket(port);
    nodeDriver.startListenerThread(socket);
    return nodeDriver;
}
}

```

The comments below apply only to multi-node API related features. See the xDB API javadocs for information on API methods used in the java code above. Each federation has a primary node, which is created at the federation creation time. An Administrator can specify primary node parameters like primary node log files directory in the `XhiveFederationIf.createFederation(...)` API method. An Administrator can also add an arbitrary number of additional nodes using the `XhiveFederationIf.addNode(...)` method. The method adds the node specification to the bootstrap file and should be run within a transaction initiated by a superuser session.

It's important to notice that each node has a separate log directory, but it is not possible to run a distributed transaction over 2 nodes and update these 2 nodes. It is possible to read libraries bound to different nodes within a transaction, but you can update only one node. Otherwise, an `XhiveException` will be thrown.

To start a node, get and initialize a driver for the node, and then start a listening thread to listen to requests from clients. If deployment includes a multi-node architecture, then use of xDB client/server mode is assumed. Embedded xDB mode is not applicable for the multi-node feature.

The `setup()` method starts the server nodes. Each server node listens for client requests on a separate port number, but a client should always connect to a primary node. This means that, to start to work with multi-node servers, a client should first create a remote driver for the primary node using `XhiveDriverFactory.getDriver(primary-node-URL)`, and then create sessions using this driver. If a client accesses a library bound to a non-primary node, then xDB automatically redirects requests to the correct node server.

Changing library binding

The following java code fragment demonstrates how to create application libraries and bind them to the server nodes according to the [deployment topology](#).

```
/**
 * Create a detachable concurrent library and bind it to the list of specified nodes.
 * @rootLib root library of the database
 * @segmentId id of the segment
 * @libName name of the new library
 * @nodeName binding node
 */
private void createLibrary(XhiveLibraryIf rootLib,
                          String segmentId, String libName, String nodeName) {
    // create segment to store new library
    rootLib.getDatabase().createSegment(segmentId, null, 0);

    // create detachable concurrent library in segment
    // by default the library is bound to the primary node
    XhiveLibraryIf library = rootLib.createLibrary(XhiveLibraryIf.DETACHABLE_LIBRARY |
                                                  XhiveLibraryIf.CONCURRENT_LIBRARY, segmentId);
    library.setName(libName);

    //append library to the root library
    rootLib.appendChild(library);

    //change binding of the library replacing primary node binding
    library.changeBinding(nodeName);
}
```

The `createLibrary(...)` method creates a detachable concurrent library in a separate segment and binds it to a server node. The `XhiveLibraryIf` interface contains more methods like `addBinding(...)`, `removeBinding(...)` and `getBindingNodes(...)` for handling library bindings properly.

Note: Only detachable libraries can be bound to non-primary server nodes.

In the event of a node failure, the Administrator can bind the standby server node to the library served by the failed node. However, this technique can only be used for read-only libraries. For a read/write library, if the node serving it fails, the only options are to either restart the same node, or for another preconfigured node to replace the failed one.

Samples

[CreateMultinodeDatabase.java](#)

Replacing a server

Various circumstances can require replacement of a host machine, for example: a need for a faster server, or a hardware failure or server crash.

In multi-node environments, replacing a host for a non-primary server will be different than for a primary server. Furthermore, replacement can involve replacing the entire node, or keeping the node and replacing only the underlying host machine. The latter case is called a *node identity change*.

For guidelines on host replacement, see

- [Changing node identity, page 314.](#)
- [Replacing a non-primary server, page 314.](#)
- [Replacing a primary server, page 315.](#)

Replacing a non-primary node

The steps below replace a non-primary server in a multi-node configuration. Node N1, running on host H1, is replaced with node N2 running on a more powerful host H2. Node server N1 is running and healthy.

To replace the non-primary node:

1. Set all the libraries that are bound to node N1 to a read-only state.
2. Install xDB server on host H2.
3. Use the **addNode()** method of the XhiveFederationIf interface to add a new node N2 on host H2 with a proper port number.
4. Use the **changeBinding()** method of the XhiveLibraryIf interface to change the binding of all libraries that are bound to node N1 to node N2.
Now node N1 is free of any binding libraries and can be removed without affecting the federation.
5. Shut down node server N1.
6. Use the **removeNode()** method of the XhiveFederationIf interface to remove node server N1 from the configuration.

Changing node identity

Replacing the underlying host machine while retaining the node name is called a *node identity change*. A node identity change is used to recover libraries when a hardware failure occurs. If libraries that are bound to the failed machine are in an inconsistent state, they must be recovered. In this case the node name must be retained while the host machine is replaced, because the recovery process reads transaction logs of the node server.

The following assumes that Node N1 is running on the non-primary host server H1 and experiences a hardware failure. Host H1 must be replaced with host H2 while the node name N1 is retained.

To replace a non-primary host while keeping the node name:

1. Shut down the node server N1, if the server is still running.
2. Ensure that the primary server is up running.
3. Install xDB server on the new host H2.
4. Use the `updateNode()` method of the `XhiveFederationIf` interface to update node N1 information in the bootstrap file. Replace host name and port number with the host name and port number of host H2. Do not change the node name N1.
5. Start node server N1 on host H2.
The server startup process recovers all libraries that are bound to node N1 to a consistent state.

Replacing a primary node

If the primary server crashes, the federation cannot be accessed because it is not possible to establish a session to the primary server. The only way to recover from a primary node failure is to change the identity of the primary node. Changing the identity requires replacing the underlying host machine on which the primary node is running while keeping the primary node name.

To replace a primary server:

1. Install xDB on the new host machine.
2. Restart the server on the new host as primary server.

It is not necessary to update the primary node. The node element for the primary node in the bootstrap file does not record host name and port number.

Example

The following code snippet launches a backend server as the primary server, by specifying the node name **primary** as the second argument in the `getDriver()` method.

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver(bsFile, "primary");
driver.init(1024);
ServerSocket socket = new ServerSocket(1235);
driver.startListenerThread(socket);
```

The primary server has a new `xhive://host:port` URL. Applications must reconnect to the primary server using the new URL.

Ant Tasks

This chapter contains the following topics:

- [Using xDB Ant tasks](#)
- [Using the `xhive.bootstrap` property with Ant](#)
- [xDB Ant type reference](#)
- [Referencing xDB Ant types](#)
- [xDB Ant task reference](#)

Using xDB Ant tasks

xDB offers custom Ant tasks and Ant types that allow you to use simple XML-based build files to perform common tasks associated with deploying and managing an xDB application, like creating and managing database structures. They can also help to reduce the need for writing java programs to solve incidental problems.

For descriptions of xDB Ant tasks, refer to the [xDB Ant task reference, page 323](#). For descriptions of xDB Ant types, refer to the [xDB Ant type reference, page 318](#).

Apache Ant is a Java-based build tool. For more information about Apache Ant, refer to <http://ant.apache.org>. xDB requires Ant 1.6 or higher. The default build file for the custom xDB Ant tasks is `build.xml` in the `bin` directory of the xDB installation.

To use the xDB Ant tasks and types in a project of your own, the build file for your project must import the xDB taskdefs and xDB typedefs, as in the example below:

```
<!-- Import all xDB jars into the classpath -->
<path id="MyClasspath">
  <fileset dir="c:/xhive/lib">
    <include name="**/*.jar"/>
  </fileset>
</path>

<!-- Load xDB Ant tasks -->
<taskdef loaderref="xhive"
  resource="com/xhive/anttasks/tasks.properties"
  classpathref="MyClasspath"/>

<!-- Load xDB Ant types -->
<typedef loaderref="xhive"
  resource="com/xhive/anttasks/type.properties"
  classpathref="MyClasspath"/>
```

The `loaderref` attribute is required here, because the loaders for the xDB tasks and types must be the same, whereas Ant uses different loaders for every task and type definition.

Unlike the command-line client of xDB, the xDB Ant tasks do not read the default values from the `xdb.properties` configuration file. You can read the properties manually and pass them to the xDB Ant tasks in your Ant build file.

The Ant tasks use Ant's built-in message logging.

Using the `xhive.bootstrap` property with Ant

With `xhive-ant`, the `xhive.bootstrap` property automatically points to the location of the page server. The syntax of the `xhive.bootstrap` property as used with `xhive-ant` is

```
java -Dxhive.bootstrap=xhive://hostname:port
```

or

```
java -Dxhive.bootstrap=PathName/XhiveDatabase.bootstrap
```

xDB Ant type reference

xDB supports various Ant types to indicate the context in which an Ant tasks acts. You can use them to indicate the federation, database, library, document, user, group or subpath that an Ant task works on.

See the descriptions of the individual Ant types for more information.

Related references

[<database/>](#)

[<document/>](#)

[<federation/>](#)

[<group/>](#)

[<library/>](#)

[<user/>](#)

[<subpath/>](#)

<database/>

This Ant type represents a database in a federation.

Attribute	Description	Required
<code>name</code>	The name of the database.	Yes
<code>bootstrap</code>	Path to bootstrap file.	Yes

Attribute	Description	Required
user	User name to log in.	Yes
password	Password of the user.	Yes

Parameters specified as nested elements

Parameter	Description	Required
<code><library/></code> , page 320	Library element - may occur multiple times.	No
<code><document/></code> , page 319	Document element - may occur multiple times.	No

Example

```
<database id="database1"
  name="MyDatabase"
  bootstrap="c:/xhive/data/XhiveDatabase.bootstrap"
  user="username"
  password="Password"/>
```

<document/>

This Ant type represents a document path in the database.

Attribute	Description	Required
path	The path to the document.	Yes

Example

```
<document id="mydocument" path="/dir/dir2/SomeDocument"/>
```

<federation/>

This Ant type represents a federation.

Attribute	Description	Required
bootstrap	Path to the bootstrap file.	Yes
password	Superuser password.	Yes
licensekey	An xDB license key.	No

Example

```
<federation id="federation1"
  bootstrap="c:/xhive/data/XhiveDatabase.bootstrap"
  password="TheSuperUserPassword"/>
```

<group/>

This Ant type represents a user group in a database.

Attribute	Description	Required
name	The name of the group.	Yes

Parameters specified as nested elements

The following optional parameters can be specified as nested elements:

Parameter	Description	Required
<user/>, page 322	User element - may occur multiple times.	No

<library/>

This Ant type represents a library path in a database.

Attribute	Description	Required
path	Library path	Yes

Parameters specified as nested elements

Parameter	Description	Required
<document/>, page 319	A document in the library - may occur multiple times.	No

Example

```
<library id="myLibrary5" path="/existingLib"/>
```

<subpath/>

This Ant type represents an **XhiveSubPathIf** instance for a multipath index. For information about multipath indexing, refer to [<multipathindex/>](#), page 351 and [Multipath indexes](#), page 153.

Attributes

Attribute	Description	Required
<code>xpath</code>	The path to the document.	Yes
<code>type</code>	Specifies the indexed element value type , page 161.	No - default is <code>string</code>
<code>fulltextindex</code>	Create a full-text index.	No - default is <code>false</code>
<code>valuecomparison</code>	Index the value of this node.	No - default is <code>false</code>
<code>includedescendants</code>	Include text from subnodes to the full-text index.	No - default is <code>false</code>
<code>startendmarkers</code>	Add start and end markers to full-text index.	No - default is <code>false</code>
<code>leadingwildcard</code>	Enable leading wildcard search in this subpath (<code>*pattern</code>).	No - default is <code>false</code>
<code>enumerateelements</code>	Enumerate the order of repeated elements.	No - default is <code>false</code>
<code>returningcontents</code>	Enable returning indexed node contents directly from index.	No - default is <code>false</code>
<code>compressed</code>	Compress part of the index.	No - default is <code>false</code>
<code>scoreboost</code>	Boost factor for the score of this subpath.	No - default is <code>1</code> .

Example

```
<subpath xpath="line"
  fulltextsearch="true"
  valuecomparison="false"
  compressed="true"
  returningcontents="true"
  includedescendants="true"
  enumerateelements="true"
  startendmarkers="true"
  leadingwildcard="true"/>
```

API documentation

[com.xhive.index.interfaces.XhiveSubPathIf](#)

<user/>

This Ant type represents a user in the database.

Attribute	Description	Required
name	The user name.	Yes
password	The user's password.	No

Parameters specified as nested elements

Parameter	Description	Required
<group/>, page 320	Group element - may occur multiple times.	No

Referencing xDB Ant types

Because the contexts used in a typical deployment build-file are likely to be used several times in different tasks, xDB makes it possible to define them in a single target. Reference them in other targets using their ID attribute. For example, change the `init` target, as described in the following example.

```
<!-- Preferably initialize your project using one target on which all
your other targets depend -->
<target name="init">

    <!-- Import all xDB jar's into the classpath -->
    <path id="MyClasspath">
        <fileset dir="lib">
            <include name="xhive-ant.jar"/>
        </fileset>
        <fileset dir="c:/xhive/lib">
            <include name="**/*.jar"/>
        </fileset>
    </path>

    <!-- Load xDB Ant tasks_ -->
    <taskdef loaderref="xhive"
        resource="com/xhive/anttasks/tasks.properties"
        classpathref="MyClasspath"/>

    <!-- Load xDB Ant types_ -->
    <typedef loaderref="xhive"
        resource="com/xhive/anttasks/type.properties"
        classpathref="MyClasspath"/>
```

```

<!-- Define a federation type. You can reference it using its "id"
attribute -->
<federation id="MyFederation"
    bootstrap="c:/xhive/data/XhiveDatabase.bootstrap"
    password="MySuperUserPassword"/>

<!-- Define a database type. You can reference it using its "id"
attribute -->
<database id="MyDatabase"
    bootstrap="c:/xhive/data/XhiveDatabase.bootstrap"
    name="MyDatabase"
    user="MyUser"
    password="MyPassword">
<library path="/MyOtherLibrary"/>
</database>
</target>

```

allows you to change the examples to:

```

<createdatabase name="MyDatabase"
    dbapassword="MyPassword">
    <federation refid="MyFederation"/>
</createdatabase>

```

or using the `databaseref` attribute:

```

<createdatabase name="MyDatabase"
    dbapassword="MyPassword">
    databaseref="MyFederation"/>

```

and to:

```

<createlibrary name="MyLibrary">
    <database refid="MyDatabase">
        <library path="/MyOtherLibrary"/>
    </database>
</createlibrary>

```

or using the `databaseref` attribute:

```

<createlibrary name="MyLibrary" databaseref="MyDatabase"/>

```

Note: Using the `databaseref` attribute allows referencing only one federation or database at a time.

xDB Ant task reference

xDB supports various Ant tasks. See the description of the individual Ant task for more information.

If you make a "compilation" like target that calls several of your targets in a particular order, remember to let this target depend on the same "init" like target the individual targets depend on. Add `inheritrefs="true"` to all `antcall` elements.

Example

```

<target name="jeroen" depends="init">
    <antcall target="test-createdatabase" inheritrefs="true"/>
    <antcall target="test-deletedatabase" inheritrefs="true"/>
</target>

```

<addgroup/>

This Ant task adds a group to a database. There are two mutually exclusive ways to use this task:

- with the name attribute, to add a single group
- with one or more nested <group/> elements, to add a number of groups

Attribute	Description	Required
name	The name of the group that is added. This attribute cannot be used in conjunction with the nested <group/> element.	No
quiet	Task progress is not displayed.	No - default is false
failonerror	Fail the task if the group already exists.	No - default is true

Parameters specified as nested elements

Parameter	Description	Required
<database/>, page 318	The database that contains the new group(s).	No
<user/>, page 322	Nested elements specifying the member(s) of a group.	No

Examples

Use of the name attribute:

```
<target name="addgroup-using-name">
  <addgroup databaseref="MyDatabase.ref" name="group1" />
</target>
```

Use of nested <group/> elements:

```
<target name="add-moregroups">
  <addgroup databaseref="MyDatabase.ref">
    <group name="group5" />
    <group name="group6" />
  </addgroup>
</target>
```

Use of nested <group/> elements with nested <user/> elements. The users are created as members of the group.

```
<target name="addgroupuser">
  <addgroup databaseref="test.database">
    <group name="group1" />
    <group name="group2">
      <user name="alice" password="secret">
        <group name="another_group" />
      </user>
    </group>
  </addgroup>
</target>
```

```

        </user>
        <user name="bob" password="secret" />
    </group>
</addgroup>
</target>

```

<addsegmentfile/>

The addsegmentfile Ant task adds a data file to a database segment.

Attribute	Description	Required
segmentid	The unique ID of the segment to which the file will be added.	Yes
path	The path to server side directory where the data file should be created. If not specified, then the path is the same as that for the federation default database.	No
maxsize	The maximum size (in bytes) that the file is allowed to grow to. Default is 0. A value of 0 means the size is unlimited.	No

Examples

The example below creates a segment "newsegment" and then adds to it a datafile (in the path and with the maxsize specified):

```

<target name="addMySegmentFiles">
    <addsegment segmentid="newsegment" databaseref="test.database" />
    <addsegmentfile segmentid="newsegment" databaseref="test.database"
        path="seg2Path" maxsize="122880" />
</target>

```

<adduser/>

This Ant task adds one or more users to a database.

There are two mutually exclusive ways to use this task:

- with the name attribute, to add a single user.
- with one or more nested <user/> elements, to add a number of users.

Attribute	Description	Required
name	The name of the user that is added. This attribute cannot be used in conjunction with the nested <user/> element.	No
password	The password of the new user. This attribute cannot be used in conjunction with the nested <user/> element.	Required in conjunction with the name attribute.
quiet	Task progress is not displayed.	No - default is false
failonerror	Fail the task if the user already exists.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<database/>, page 318	The database that contains the new user(s).	No.
<user/>, page 322	A user that is created.	No.

Examples

Use of the name attribute:

```
<target name="adduser-target">
  <adduser databaseref="MyDatabase.ref" name="jeroen" password="secret" />
</target>
```

Use of nested <user/> elements:

```
<target name="add-many-users">
  <adduser databaseref="MyDatabase.ref">
    <user name="alice" password="secret" />
    <user name="bob" password="secret" />
  </adduser>
</target>
```

<backup/>

This Ant task creates an online (hot) backup of the federation. This requires that the database server is running.

Attribute	Description	Required
file	The file that stores the database backup.	Yes
overwrite	If the file already exists, overwrite it.	No - default is false

Attribute	Description	Required
incremental	Create an incremental backup that includes only the log files since the latest, non-standalone backup.	No - default is false
keeplogfiles	Do not remove obsolete log files after the backup.	No - default is false
standalone	Create a standalone backup. This option implies the keeplogfiles attribute. A standalone backup does not interrupt the sequence of incremental backups. Incremental backups cannot be created with respect to a standalone backup. Only one backup that is not standalone can be created at the same time.	No - default is false
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
database	The database to back up.	No

Example

The following example passes a reference to a federation as a database reference.

```
<target name="make-a-backup">
  <backup databaseref="MyFederation.ref" file="new_backup.db" />
</target>
```

<batchindexadder/>

This Ant task adds multiple indexes in one batch operation, using the **XhiveIndexAdderIf** interface. For general information about indexing, refer to [Indexes](#), page 150.

Attribute	Description	Required
file	XML Document containing exported index definitions.	No

Parameters specified as nested elements

Index	Description
<database/>	Specifies a database and library where the index(es) must be added. For the library, use a nested <library/> element (see example below).
<pathvalueindex/>	Add a path value index.
<multipathindex/>	Add a multi path index.

Index	Description
<elementindex/>	Add an element index.
<fulltextindex/>	Add a full-text index.
<idattributeindex/>	Add an ID attribute index.
<libraryidindex/>	Add a library index.
<metadatavalueindex/>	Add a metadata full-text index.
<metadatavalueindex/>	Add a metadata value index.
<valueindex/>	Add a value index.

Example

```
<target name="add-indexes">
  <batchindexadder>
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
    <metadatavalueindex name="MyMVIndex"
      key="bla"/>
    <valueindex name="MyValueIndex" elementURI="http://www.x-hive.com/ns"
      elementName="title"/>
    ...
  </batchindexadder>
</target>
```

API documentation

com.xhive.index.interfaces.XhiveIndexAdderIf.html

<checkdatabase/>

This task uses the checkDatabaseConsistency API call to check the consistency of a database.

Parameters

Parameter	Description	Required
<database/>	Nested element specifying the database to be checked.	Yes
Checkdomnodes	Check the dom nodes.	No - default is true
CheckIndexes	Check the indexes.	No - default is true
CheckAdministrationPages	Check the administration pages.	No - default is true
CheckSegmentPages	Check the segment pages.	No - default is true
CheckPageOwner	Check the pages owner.	No - default is true

Example

The example below checks the consistency of a database.

```
<target name="checkdatabase-target">
  <checkdatabase destination="NewName">
    <database bootstrap="MyBootstrapFile"
      name="MyDatabase"
      user="Administrator"
      password="MyPassword"
      Checkdomnodes="false"
      CheckIndexes="false"
      BasicCheckIndexes="true"
      CheckAdministrationPages="false"
      CheckSegmentPages="true"
      CheckPageOwner="false"/>
  </checkdatabase>
</target>
```

API documentation

[com.xhive.index.interfaces.XhiveConsistencyCheckerIf.html#checkDatabaseConsistency\(\)](http://com.xhive.index.interfaces.XhiveConsistencyCheckerIf.html#checkDatabaseConsistency())

<checkfederation/>

This task uses the **checkFederationConsistency** API call to check the consistency of a federation.

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<federation/>	Nested element specifying the federation.	Yes
Checkdomnodes	Check the DOM nodes.	No - default is true
CheckIndexes	Check the indexes.	No - default is true
CheckAdministrationPages	Check the administration pages.	No - default is true
CheckSegmentPages	Check the segment pages.	No - default is true
CheckPageOwner	Check the pages owner.	No - default is true

Example

The following code example checks the consistency of a federation.

```
<target name="checkfederation-target">
  <checkfederation>
    <federation refid="test.federation"/>
  </checkfederation>
</target>
```

```

    </checkfederation>
</target>

```

API documentation

[com.xhive.index.interfaces.XhiveFederationConsistencyCheckerIf.html#checkFederationConsistency\(\)](http://com.xhive.index.interfaces.XhiveFederationConsistencyCheckerIf.html#checkFederationConsistency())

<checklibrarychild/>

This Ant task checks the consistency of a library child.

This task uses the `XhiveConsistencyCheckerIf.checkLibraryChildConsistency` API call.

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database which contains the library.	Yes.
name	The root path of the library to be checked	No
<library/>	The library to be checked under root path.	No.
Checkdomnodes	Check the DOM nodes.	No - default is true
CheckIndexes	Check the indexes.	No - default is true
CheckAdministrationPages	Check the administration pages.	No - default is true
CheckSegmentPages	Check the segment pages.	No - default is true
CheckPageOwner	Check the pages owner.	No - default is true

Example

The following code example checks the consistency of a library.

```

<target name="checklibrary-target">
  <checklibrarychild name="testLib"
    Checkdomnodes="false"
    CheckIndexes="false"
    BasicCheckIndexes="true"
    CheckAdministrationPages="false"
    CheckSegmentPages="true"
    CheckPageOwner="false">

```

```

    <database bootstrap="MyBootstrapFile"
      name="MyDatabase"
      user="Administrator"
      password="MyPassword" >
      <library path="/existingLib" />
    </database>
  </checklibrarychild>
</target>

```

<checknode/>

This Ant task checks the consistency of a node.

Parameters

This task uses the API call `XhiveFederationConsistencyCheckerIf.checkNodeConsistency`.

The following optional parameters can be specified as nested elements:

Parameter	Description	Required
node	The node name to be checked	Yes
<federation/>	The federation the node belongs to.	Yes
Checkdomnodes	Check the DOM nodes.	No - default is true
CheckIndexes	Check the indexes.	No - default is true
CheckAdministrationPages	Check the administration pages.	No - default is true
CheckSegmentPages	Check the segment pages.	No - default is true
CheckPageOwner	Check the pages owner.	No - default is true

Example

The example below checks the consistency of a node.

```

<target name="checknode-target">
  <checknode node="nodeA">
    <federation refid="test.federation"/>
  </checknode>
</target>

```

<closedriver/>

This Ant task closes a federation driver. **Note:** The driver is also closed when the JVM of the Ant process exits.

Other xDB Ant tasks do not close the XhiveDriver after they run, to avoid the performance overhead associated with closing and opening a driver between subsequent xDB tasks. This can become a

problem if you wish to use xDB Ant tasks, and spawn a new process that will use the database inside the same Ant process. For example, deploying a servlet using the federation driver on Tomcat.

Note:

Attribute	Description	Required
bootstrap	The path to the federation bootstrap file.	Yes

Example

The example below closes a driver.

```
<target name="deploy-webdav">
  <!-- create libraries and add users etc -->
  <addlibrary etc />
  <adduser etc />

  <!-- close the driver -->
  <closedriver bootstrap="\${xhive.bootFilePath}"/>

  <!-- proceed to deploy xDB Webdav module on Tomcat -->
</target>
```

<copydatabase/>

This Ant task copies a federation database.

If the destination for the copy of the database exists, the task raises an `XhiveException.DATABASE_EXISTS`.

Note: The copy process does not copy empty pages, so the copy of the database may be smaller than the original. Content conditioned indexes are **not** copied; only their definition remains.

Attribute	Description	Required
name	The database that is copied.	Yes
destination	The name of the new database copy.	Yes
sourcepassword	The administrator password of the original database.	Yes
supassword	The superuser password.	Yes
quiet	Do not display task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
federation	The federation containing the databases.	No.

Example

The example below copies a database.

```
<target name="test-copydatabase">
  <copydatabase name="MyDatabase"
    destination="NewName"
    sourcepassword="MyPassword"
    supassword="SuperUserPassword">
    <federation refid="MyFederation" />
  </copydatabase>
</target>
```

<createdatabase/>

This Ant task creates a federation database with a default configuration. If a database with the same name already exists, the Ant execution script will fail unless failonerror is set to false.

Attribute	Description	Required
name	The name of the database to be created.	Yes
dbapassword	The administrator password for the new database.	Yes
configuration	XML file containing a custom configuration for the newly created database.	No
quiet	Do not display task progress.	No - default is false
failonerror	Fail the task if the database already exists.	No - default is true

Parameters specified as nested elements

Parameter	Description	Required
<federation/>	The federation in which the database will be created.	Yes

Example

The example below creates a database.

```
<!-- Declare a federation with an ID, which is used in the database creation task. -->
<federation id="federationId" bootstrap="${xhive.bootFilePath}"
  password="${xhive.superpwd}" />

<target name="create-database">
  <createdatabase name="${database}" dbapassword="${password}">
    <federation refid="federationId" />
  </createdatabase>
</target>
```

<createfederation/>

This Ant task creates a federation. The bootstrap file, superuser password and license key can be set explicitly by inserting a [<federation/> element, page 319](#).

Attribute	Description	Required
bootstrap	The name of the bootstrap file.	Yes
password	The password of the superuser.	Yes
licensekey	The xDB license key.	Yes
pagesize	The database page size in bytes.	No
logdir	A comma separated list of paths to the log file directories of the new federation. If specified, the first path in the list represents the primary log directory. Relative paths, if used, are resolved relative to the directory of the bootstrap file.	No - default path is log
quiet	Do not display task progress.	No - default is false

Example

The example below creates a federation.

```
<target name="create-federation">
  <delete dir="MyBootstrapFile" />
  <mkdir dir="MyBootstrapFile" />
  <createfederation bootstrap="MyBootstrapFile"
    licensekey="MyLicenseKey"
    password="secret" />
</target>
```

<createlibrary/>

This Ant task creates a library in a database.

Attribute	Description	Required
name	The name of the new library. The library name must be unique. If a library with the same name already exists, this Ant task is ignored.	Yes
documentslock	Documents in the new library lock with the parent.	No - default is true
lockwithparent	The new library locks with its parent.	No - default is false
concurrentlibrary	The new library can be modified concurrently.	No - default is false
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<database/>, page 318	The database where the new library is created.	No

Example

The following code example creates a library.

```
<target name="CreateMyLibraryUnderParent" depends="init">
  <createlibrary name="MyLibrary">
    <database refid="MyDatabase">
      <library path="/MyOtherLibrary"/>
    </database>
  </createlibrary>
</target>
```

<deletedatabase/>

This Ant task deletes a database in a federation. If a database with the given name does not exist, Ant stops execution, unless the quiet attribute is set to **true**, or the failonerror attribute is set to **false**.

Attribute	Description	Required
name	The name of the database to be deleted.	Yes
failonerror	Stop execution if database name does not exist.	No - default is true
quiet	Suppress display of task progress.	No - default is true

Parameters specified as nested elements

Parameter	Description	Required
<federation/>	The federation from which to delete the database.	No

Example

The example below deletes a database.

```
<target name="DeleteMyDatabase" depends="init">
  <deletedatabase name="MyDatabase">
    <federation refid="MyFederation"/>
  </deletedatabase>
</target>
```

<deletegroup/>

This Ant task deletes one or more groups from a database.

There are two mutually exclusive ways to use this task:

- Using a name attribute to add a single group.
- Using one or more nested <group/> elements to delete a number of groups.

Attributes

Attribute	Description	Required
name	The name of group to delete. The name attribute cannot be used in conjunction with nested <group/> elements.	Yes.
failonerror	Specifies whether execution should stop when a database with the name does not exist. The default value is true .	No.
quiet	Specifies whether output about the task progress should be displayed. The default value is false .	No.

Parameters

The following optional parameter can be specified as nested elements:

Parameter	Description	Required
<database/>	The database that contains the group that is deleted.	No.
<user/>	The group to delete.	No.

Example

The following example uses the name attribute to delete a group:

```
<target name="delete-one-group">
```



```

    <deletegroup databaseref="MyDatabase.ref" name="group1" />
</target>

```

Using a nested `<group/>` elements to delete one or more groups.

```

<target name="delete-many-groups">
  <deletegroup databaseref="test.database">
    <group name="jeroen3_group" />
    <group name="jeroen4_group" />
  </deletegroup>
</target>

```

<deleteindex/>

This Ant task deletes an index.

Attribute	Description	Required
name	The name of the index to delete.	Yes
failonerror	Stop execution if the specified name does not exist.	No - default is true
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<database/> , page 318	The database containing the library where the index is deleted.	No.

Example

The example below uses a nested `<database/>` with a nested `<library/>` element.

```

<target name="DeleteMyIndex" depends="init">
  <deleteindex name="MyIndex">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </deleteindex>
</target>

```

<deletelibrary/>

This Ant task deletes a library. If the specified library does not exist, execution stops, unless the quiet attribute is set to true or the failonerror attribute is set to false.

Attribute	Description	Required
name	The name of the library to delete.	No
	This attribute cannot be used together with "path".	

Attribute	Description	Required
path	The full path of the library to delete. This attribute cannot be used together with "name".	No
failonerror	Stop the Ant task if the library does not exist.	No - default is true
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database from which to delete the library.	No

Example using the attribute "name"

```
<target name="DeleteMyLibrary" depends="init">
  <!-- Deletes "/MyLibrary/document.xml" -->
  <deletelibrary name="document.xml">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </deletelibrary>
  <!-- Deletes "/MyOtherLibrary/MyLibrary2" -->
  <deletelibrary name="MyLibrary2">
    <database refid="MyDatabase">
      <library path="/MyOtherLibrary"/>
    </database>
  </deletelibrary>
</target>
```

Example using the attribute "path"

```
<target name="DeleteMyLibrary"
depends="init">
  <!-- Deletes "/MyLib/MyOtherLib/document.xml" -->
  <deletelibrary path="/MyLib/MyOtherLib/document.xml">
    <database refid="MyDatabase" />
  </deletelibrary>
</target>
```

<deleteuser/>

This Ant task deletes one or more users from a database. There are two mutually exclusive ways to use this task:

- Using a name attribute to delete a single user.
- Using one or more nested `<user/>` elements to delete a number of users.

Attribute	Description	Required
name	The name of a user to delete. The name attribute cannot be used in conjunction with nested <user/> elements.	Yes.
failonerror	Stop execution if the user does not exist.	No - default is true
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<database/>, page 318	The database containing the users to delete.	No
<user/>, page 322	A user to delete. If the <deleteuser/> Ant task specifies a name attribute, the <user/> parameter is ignored.	No

Example

Using the name attribute to delete a single user:

```
<target name="delete-user-target">
  <deleteuser databaseref="MyDatabase.ref" name="jeroen" failonerror="false"/>
</target>
```

Using <user/> elements to delete users:

```
<target name="delete-many-users">
  <deleteuser databaseref="MyDatabase.ref" failonerror="true">
    <user name="alice" />
    <user name="bob" />
  </deleteuser>
</target>
```

<deserialize/>

This Ant task deserializes a library child from a specified file. The library child in the source file becomes the last child of the target library.

Note: If no target library is specified, the deserialized library replaces the current root library of the database.

Attributes

Attribute	Description	Required
file	The source file to deserialize.	Yes
quiet	Suppress display of task output.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database to contain the deserialized library.	No

Example

The following example deserializes the content of a file named MyLibrary.xhd.

```
<target name="DeserializeMyLibrary" depends="init">
  <deserialize file="c:/MyLibrary.xhd">
    <database refid="MyDatabase">
      <library path="/" />
    </database>
  </deserialize>
</target>
```

`<deserialize-users/>`

This Ant task deserializes all users and groups of a database, replacing the current users and groups.

Attribute	Description	Required
file	The source file to deserialize.	Yes

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database into which to deserialize the users and groups.	No

Example

The following example deserializes the contents of the file MyLibrary.xhd.

```
<target name="DeserializeUsers" depends="init">
  <deserialize-users file="c:/MyLibrary.xhd">
    <database refid="MyDatabase" />
  </deserialize-users>
</target>
```

<elementindex/>

This Ant task adds an element index to a library.

Attribute	Description	Required
name	The name of the index that is added.	Yes
elements	A comma separated list of element names to index.	No
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database containing the library where the index is added.	No

Example

The example below contains a nested `<database/>` element with a nested `<library/>` element.

```
<target name="AddMyElementIndex" depends="init">
  <elementindex name="MyElementIndex">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </elementindex>
</target>
```

<exportlibrary/>

This Ant task exports a library into a database directory.

Attributes

Attribute	Description	Required
destdir	The destination directory to which to export the library. By default, the current directory is the export directory.	No
prune	Specifies whether to create directories for empty libraries. The default is true .	No

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to export.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="ExportMyLibrary" depends="init">
  <exportlibrary destdir="c:/exportdata">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </exportlibrary>
</target>
```

<fulltextindex/>

This Ant task adds a value full-text index to a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes
elementURI	The URI of the element to index.	Yes
elementName	The name of the element to index.	This attribute is required if the attributeName attribute value is null .
attributeURI	The URI of the attribute to index.	Yes
attributeName	The name of the attribute to index.	Required if the value of the elementName attribute value is null .

Attribute	Description	Required
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
alltext	Boolean attribute that specifies whether indexed nodes can contain children, indexing the complete text of all nodes.	No - default is false
analyzer	The analyzer class name.	No
supportphrases	Boolean attribute that specifies whether to optimize the index to perform phrase queries. This option improves the quality of scoring results.	No - default is true
supportscoring	Boolean attribute that specifies whether the use of scoring is supported.	No - default is true
lowercase	Boolean attribute that specifies whether indexed terms are converted to lower case. When this option is used, queries are not case-sensitive.	No - default is true
stopwords	Boolean attribute that specifies whether words are not indexed if they are from a list of standard English stopwords.	No - default is true
leadingwildcard	Boolean attribute that specifies whether to support efficient searches with a leading wildcard, such as in "*TERM". Using this option can increase the size of the index considerably.	No - default is false
supportprefixwildcard	Deprecated name for leadingwildcard. Using this option can increase the size of the index considerably.	No - default is false
includeattributes	Boolean attribute that specifies whether to index the attribute text of indexed elements.	No - default is false

Attribute	Description	Required
includestartendtokens	Boolean attribute that specifies whether to include index support for "at start" and "at end" XQFT filters.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyFulltextIndex" depends="init">
  <fulltextindex name="MyFullTextIndex"
    elementName="title"
    alltext="true">
    <database refid="databaseRef">
      <library path="/MyLibrary/SubLib"/>
    </database>
  </fulltextindex>
</target>
```

<idattributeindex/>

This Ant task adds an ID attribute index on a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes

Attribute	Description	Required
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
unique	Boolean attribute that specifies whether to use unique keys.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyIDattributeIndex" depends="init">
  <idattributeindex name="MyIDattributeIndex">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </idattributeindex>
</target>
```

<libraryidindex/>

This Ant task adds a library ID index to a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes

Attribute	Description	Required
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyLibraryIDindex" depends="init">
  <libraryidindex name="MyLibraryIDindex">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </libraryidindex>
</target>
```

<listindexes/>

This Ant task lists all indexes for a database library path.

Attributes

Attribute	Description	Required
definitionsToXml	String attribute specifying a file to which export the definitions of all indexes in the given library. If no value is given the definitions are not exported.	No
info	Boolean attribute that specifies whether to provides additional information about each index. The default value is false .	No

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library. If the database element does not include any nested <library/> elements, the task lists the indexes at the root library of the database.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="list-indexes">
  <listindexes info="true">
    <database refid="MyDatabase.ref">
      <library path="path/to/SomeLibrary"/>
      <library path="anotherLibrary"/>
    </database>
  </listindexes>
</target>
```

<metadatafulltextindex/>

This Ant task adds a metadata full-text index to a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes
key	The name of the metadata field to index.	Yes
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false

Attribute	Description	Required
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
supportphrases	Boolean attribute that specifies whether to optimize the index to perform phrase queries.	No - default is true
leadingwildcard	Boolean attribute that specifies whether to support efficient searches with a leading wildcard, such as in "*TERM". Using this option can increase the size of the index considerably.	No - default is false
supportprefixwildcard	Deprecated name for leadingwildcard. Using this option can increase the size of the index considerably.	No - default is false
supportscoring	Boolean attribute that specifies whether the use of scoring is supported.	No - default is true
analyzer	The analyzer class name.	No - default is none
lowercase	Boolean attribute that specifies whether to convert the indexed terms to lower case. This option only applies if the <code>analyzer</code> is left at its default value. Queries are not case-sensitive.	No - default is true
stopwords	Boolean attribute that specifies whether words are not indexed if they are from a list of standard English stopwords. This option only applies if the <code>analyzer</code> is left at its default value.	No - default is true
includestartendtokens	Boolean attribute that specifies whether to include index support for "at start" and "at end" XQFT filters.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested `<database/>` and a nested `<library/>` element.

```
<target name="AddMyMFTIndex">
  <metadatafulltextindex name="MyMFTIndex"
    key="bla">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </metadatafulltextindex>
</target>
```

<metadatavalueindex/>

This Ant task adds a metadata value index to a library.

Attributes

Attribute	Description	Required
name	The name of the index.	Yes
key	The name of the metadata field that is indexed.	Yes
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
unique	Boolean attribute that specifies whether to use unique keys.	No - default is false
valuetype	Specifies the indexed key value type, page 161 .	The default is string .
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyMVIndex">
  <metadatavalueindex name="MyMVIndex"
    key="bla">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </metadatavalueindex>
</target>
```

<metadata/>

This Ant task can set or unset metadata (**XhiveMetadataIf**) on a document, library or BLOB.

Attributes

Attribute	Description	Required
path	The path of the library child whose metadata is going to be modified.	Yes.
key	The metadata key.	Yes.
value	The new value of the metadata key.	No.
delete	Whether to delete the metadata key or not.	No. This attribute can only be set to true or false
failonerror	Specifies whether execution should stop when an index with the specified name does not exist. The default value is true .	No.

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library from which the index is deleted.	No.

Example

The following example change the metadata of the library child at path `"/myLib/MyOtherLib/document.xml"` by setting the metadata field "sauce" to the value "pomodoro".

```
<metadata key="sauce" value="pomodoro" path="/myLib/MyOtherLib/document.xml">
  <database refid="test.database" />
</metadata>
```

<multipathindex/>

This Ant task adds a multipath index to a library child. For information about multipath indexes, refer to [Multipath indexes, page 153](#)

Attribute	Description	Required
name	The name of the index that is added.	Yes
path	The main path to be indexed. All sub-paths are made relative to this one.	Yes
scorecustomizer	The class name of an implementation of XhiveScoreCustomizerIf .	No - default is none
analyzer	The analyzer class name.	No - default is none
lowercase	If the attribute <code>analyzer</code> is not set, this attribute specifies whether to convert the indexed terms to lower case.	No - default is true
stopwords	If the attribute <code>analyzer</code> is not set, this attribute specifies not to index words that are in a list of standard English stopwords.	No - default is true
exists	What to do if an index with the same name already exists. Accepted values are: <code>skip</code> - do not create a new index, <code>overwrite</code> - delete the existing index with the same name and create the newly specified index, <code>fail</code> - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database that contains the library where the multipath index is added.	No
<code><subpath/></code> , page 321	Subpath configuration for this multipath index. If no <code><subpath/></code> is included, a default subpath on <code>"//*"</code> with full text search enabled is used.	No

Example

The example below contains two nested `<subpath/>` elements, a nested `<database/>` and a nested `<library/>` element.

```
<multipathindex name="my-multipath-index"
  path="/mainXPath"
  analyzer="com.emc.textanalysis"
  scorecustomizer="com.emc.scorecustomize"
  lowercase="false"
  stopwords="false">
  <database refid="databaseRef">
    <library path="/existingLib" />
  </database>
  <subpath xpath="line"
    compressed="true"
    returningcontents="true"
    getalltext="true"
    enumerateelements="true"
    startendmarkers="true"
    leadingwildcard="true"
    fulltextsearch="true"
    valuecomparison="true" />
    <subpath xpath="bar" type="int" scoreboost=".5" valuecomparison="true" />
</multipathindex>
```

`<parse/>`

This Ant task parses files into a library. Include the `<fileset/>` element to indicate which files to parse. The parse task copies the directory structure as a library structure into the target library, unless the `flatten` attribute is set to **true**.

Attributes

Attribute	Description	Required
<code>usenamespaces</code>	Boolean attribute that specifies whether to parse namespaces.	No - default is true

Attribute	Description	Required
validate	Boolean attribute that specifies whether to validate the files that are parsed.	No - default is false
namequery	Specifies the string representing XQuery to select the element containing the name for the new database.	No
overwrite	Specifies whether to overwrite the document if it already exists. This attribute can have the following values: <ul style="list-style-type: none"> • true The existing document is overwritten. • false The existing document is not overwritten and the Ant task stops. • newer The existing document in the library is overwritten if the parsed document is newer. 	No - default is newer
flatten	Boolean attribute that specifies whether to flatten the directory structure.	No - default is false
documentslock	Boolean attribute that specifies whether documents in the newly created libraries should lock with their parent. This attribute is only relevant when the flatten attribute is set to false .	No - default is true
lockwithparent	Boolean attribute that specifies whether the newly created libraries should lock with their parents. This attribute is only relevant when the flatten attribute is set to false .	No - default is false
quiet	Boolean attribute that specifies whether the task progress is displayed.	No - default is false

Parameters

The following optional parameter can be specified as nested elements:

Parameter	Description	Required
<database/>	The database to which the library with the parsed files is added.	No
<fileset/>	The set of files to parse and add to the library.	No

Example

```
<target name="ParseInMyLibrary" depends="init">
  <parse>
    <fileset dir="data">
      <include name="**/*.xml"/>
    </fileset>
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </parse>
</target>
```

<pathvalueindex/>

This Ant task adds a path value index to a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes
path	The index path.	Yes
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
unique	Boolean attribute that specifies whether to use unique keys.	No - default is false
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element. The index is created with path `"/foo/bar[@x<INT>]". To insert literal < and > characters into an Ant build file, use the < > notation.`

```
<target name="create-path-index" depends="init">
  <pathvalueindex name="MyPathValueIndex" path="/foo/bar[@x<INT>]">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </pathvalueindex>
</target>
```

```
</target>
```

<registerreplicator/>

This Ant task registers a replicator in a federation.

Attributes

Attribute	Description	Required
name	The replicator name to register at the federation.	Yes

Parameters

The following optional parameter can be specified as nested elements:

Parameter	Description	Required
<federation/>	The federation where the replicator is registered.	No

Example

The following example registers a replicator.

```
<target name="register-MyReplicator">
  <registerreplicator name="MyReplicator">
    <federation refid="test.federation"/>
  </registerreplicator>
</target>
```

<renamedatabase/>

This Ant task renames a database in a federation.

This task uses the **XhiveDatabaseIf.renameDatabase** API call and renames the database but not the database files. A database can be renamed safely by first using the <copydatabase/> task and later deleting the original database.

Attributes

Attribute	Description	Required
destination	The name of the new database.	Yes
quiet	Boolean attribute that specifies whether the task progress is displayed. The default value is false .	No

Parameters

The following optional parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The name of the database to be renamed.	No

Example

```
<target name="renamedatabase-target">
  <renamedatabase destination="NewName">
    <database bootstrap="MyBootstrapFile"
      name="MyDatabase"
      user="Administrator"
      password="MyPassword" />
  </renamedatabase>
</target>
```

<replicatefederation/>

This Ant task replicates the whole federation. This task only performs an initial duplication. It is a simultaneous standalone backup and a restore process.

In order to move the federation to a new location, change the location of the bootstrap file. Set the `relativepath` attribute set to **true**, because all paths in the original federation are first made relative and then set to the directory of the new bootstrap file.

Attributes

Attribute	Description	Required
bootstrap	The path to the new bootstrap file. By default, the path of the original federation is used. Any relative paths in the original bootstrap file are interpreted as relative to the new bootstrap file.	Yes - default is null
relativepath	Specifies whether all paths in the federation are relative. By default, the original paths stored in the bootstrap file are used. This task can be used to restore the federation to directories different from the original location.	Yes - default is null
quiet	Boolean attribute that specifies whether the task progress is displayed.	No - default is false

Parameters

The following optional parameters can be specified as nested elements:

Parameter	Description	Required
<federation/>	The federation that is replicated.	No

<restore/>

This Ant task restores a federation from a backup. The <restore/> task does not overwrite existing files. To restore incremental backups, this task must be used to restore the last full backup first, then for each incremental backup in the order they have been created. Do not restart the server during the restore procedure.

Attributes

Attribute	Description	Required
file	The file containing the backups.	Yes
bootstrap	The name of the bootstrap file to restore the backup. If no other path is specified, the federations is restored to the same location from which it was backed up. This task can be used to restore a federation to a different location or to make a copy of a federation. Any relative paths in the original bootstrap file are interpreted as relative to the new bootstrap file.	No - default is null
relativepath	Boolean attribute that specifies that all paths in the federation are relative. If set to false , xDB uses the original paths that are stored in the bootstrap file. This task can be used to restore the federation to directories different from the original location.	No - default is false
quiet	Boolean attribute that specifies whether the task progress output is displayed.	No - default is false

Example

```
<target name="restore-mybackup">
  <restore file="lastBackup.db" bootstrap="MyBootstrapFile" />
</target>
```

<serialize/>

This Ant task serializes a library child into an output file.

Attribute	Description	Required
file	The destination file for the serialized data.	Yes
quiet	Suppress display of task progress.	No - default is false

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database with the nested elements to serialize.	No

Example

The following example serializes data from the MyDatabase database into the file MyLibrary.xhd.

```
<target name="SerializeMyLibrary" depends="init">
  <serialize file="c:/MyLibrary.xhd">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </serialize>
</target>
```

<serialize-users/>

This Ant task serializes all users and groups of a database.

Attribute	Description	Required
file	The destination file for the serialized users and groups.	Yes

Parameters specified as nested elements

Parameter	Description	Required
<code><database/></code> , page 318	The database containing the users and groups to serialize.	No

Example

The following example serializes the contents of the file MyLibrary.xhd.

```
<target name="SerializeUsers" depends="init">
  <serialize-users file="c:/MyLibrary.xhd">
    <database refid="MyDatabase"/>
  </serialize-users>
</target>
```

<session/>

The session Ant task is a container task that can contain other Ant tasks. The nested tasks are executed inside a single XhiveSession.

Attributes

Attribute	Description	Required
databaseref	The reference ID of a database or federation element. This parameter can also be provided as a nested element.	Yes

Examples

The <session/> task can contain any other xDB Ant tasks specified as nested elements. The nested Ant tasks cannot be a combination of Ant tasks that require superuser permissions and Ant tasks that do not require superuser permission. Ant tasks operating at the federation level require a nested <federation/> element.

```
<target name="use-session" depends="init">
  <session databaseref="database.RefId">
    <createlibrary name="testLib"/>
    <createlibrary name="testLib2"/>
  </session>
</target>

<target name="use-session" depends="init">
  <session>
    <database refid="test.database"/>
    <createlibrary name="testLib3" />
    <createlibrary name="testLib4" />
  </session>
</target>

<target name="test-session-fed">
  <session databaseref="federation.RefId">
    <createdatabase name="MyDB1" dbapassword="{password}"/>
    <createdatabase name="MyDB2" dbapassword="{password}"/>
    <createdatabase name="MyDB3" dbapassword="{password}"/>
  </session>
</target>
```

<setmaxfilesize/>

The setmaxfilesize Ant task sets the maximum size of a segment data file.

Attribute	Description	Required
path	The full path (case sensitive) of the data file (as produced by the show-segment command).	Yes
maxsize	The maximum size (in bytes) that the file is allowed to grow to. A value of 0 means the size is unlimited.	Yes
segmentid	The id of the segment to which the data file belongs.	No

Example

The example below sets the max file size of the newly created data file to 200000 bytes:

```
<target name="setFileMyMaxSize">
  <addsegment segmentid="newsegment" databaseref="test.database" />
  <addsegmentfile segmentid="newsegment" databaseref="test.database"
    maxsize="0" /> <property name="segment.fullname"
    location="..${file.separator}data${file.separator}MyDatabase2-newsegm
  <setmaxfilesize segmentid="newsegment" databaseref="test.database" path="${segment.fullname}
</target>
```

<unregisterreplicator/>

This Ant task cancels a replicator registration in a federation. Log files are no longer preserved for this replicator.

Attributes

Attribute	Description	Required
name	The name of the replicator for which the registration is canceled.	Yes

Parameters

The following optional parameter can be specified as nested elements:

Parameter	Description	Required
<federation/>	The federation where the replicator registration is cancelled.	No

Example

The following example cancels a replicator registration.

```
<target name="unregister-MyReplicator">
  <unregisterreplicator name="MyReplicator">
    <federation refid="test.federation"/>
  </unregisterreplicator>
</target>
```

<updatefederation/>

This Ant task updates the xDB license key of a federation. The updatefederation task closes the xDB driver.

Attributes

Attribute	Description	Required
bootstrap	The name of the the bootstrap file.	Yes
password	The password of the superuser.	Yes
licensekey	The new xDB license key.	Yes
quiet	Boolean attribute that specifies whether the task progress is displayed. The default value is false .	No

Example

The following example updates the license key in the bootstrap file.

```
<target name="update-federation">
  <updatefederation bootstrap="MyBootstrapFile"
    password="secret"
    licensekey="MyLicenseKey"/>
</target>
```

<upload/>

This Ant task uploads files into a library. To indicate which files to upload, an Ant fileset element must be included. This task can upload DOM Documents as well as BLOBs into the database. The task copies the directory structure as a library structure in the target library unless the flatten attribute is set to **true**.

Attributes

Attribute	Description	Required
xmlextensions	A comma-separated list of extensions that identify the XML files to parse. The default file extension is XML.	No
usenamespaces	Boolean attribute that specifies whether to parse namespaces.	No - default is true
validate	Boolean attribute that specifies whether to validate the files that are uploaded.	No - default is false
psvi	Boolean attribute that specifies whether to store PSVI (Post Schema Validation Infoset) after validation of XML Documents.	No - default is false
flatten	Boolean attribute that specifies whether to flatten the directory structure.	No - default is false
quiet	Boolean attribute that specifies whether the task progress is displayed.	No - default is false

Parameters

The following optional parameter can be specified as nested elements:

Parameter	Description	Required
<database/>	The database to which the library with the parsed files is added.	No
<fileset/>	The set of files to parse and add to the library.	No

Example

```
<upload xmlextensions="xml,xhtml">
  <fileset dir="${data.dir}">
    <include name="fbooks/*" />
  </fileset>
  <database refid="MyDatabaseRef">
    <library path="/testLib" />
  </database>
</upload>
```

<valueindex/>

This Ant task adds a value index to a library.

Attributes

Attribute	Description	Required
name	The name of the index that is added.	Yes
elementURI	The URI of the element to index.	Yes
elementName	The name of the element to index.	This attribute is required if the attributeName attribute value is null .
attributeURI	The URI of the attribute to index.	Yes
attributeName	The name of the attribute to index.	Required if the value of the elementName attribute value is null .
concurrent	Boolean attribute that specifies whether to create a concurrent index.	No - default is false

Attribute	Description	Required
compressed	Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes.	No - default is false
unique	Boolean attribute that specifies whether to use unique keys.	No - default is false
valuetype	Specifies the indexed element type. The default value is string . The valuetype attribute can have the value string, int, long, double, float, date, date_time, time, day_time_duration , and year_month_duration .	No
exists	What to do if an index with the same name already exists. Accepted values are: skip - do not create a new index, overwrite - delete the existing index with the same name and create the newly specified index, fail - fail the Ant task.	No - default is skip
quiet	Specifies whether to display task progress.	No - default is false
versioninfo	Boolean attribute that specifies whether to store version information in the index.	No - default is false

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<database/>	The database containing the library to which the index is added.	No

Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyValueIndex" depends="init">
  <valueindex name="MyValueIndex"
    elementURI="http://www.x-hive.com/ns"
    elementName="title">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </valueindex>
</target>
```

<xquery/>

This Ant task executes an XQuery in the context of a library. The query results can be stored in an Ant property, written to a file, or logged in the Ant build. The query must be given using a nested `<query/>` element, XQuery external variables can be set using nested `<param/>`s.

Attributes

Attribute	Description	Required
outputfile	A file to write the result of the XQuery to.	No
outputproperty	A property to store the XQuery result in. By default (if neither outputfile nor outputproperty is specified), query results are logged in the Ant build.	No

Parameters

The following parameters can be specified as nested elements:

Parameter	Description	Required
<code><database/></code>	The database containing the library to which the index is added.	Yes
<code><query/></code>	The query to run, either specified as text within the <code><query/></code> element, or as a file referenced through the file attribute. Ant properties in the element content will be expanded. Special XML characters need to be escaped, it is recommended to wrap element contents in a CDATA section (see example below).	Yes
<code><param/></code>	A parameter to declare variable in the XQuery. Attributes namespace, name, and value. Ant properties will be expanded in the value.	No

Example

The following example runs an xquery wrapped in a CDATA section, and stores the result in a property. The xquery takes an external variable (`$addressee`), which is supplied with a `<param/>` element, which in turn uses an Ant property. The library is specified by a nested `<database/>` element with a nested `<library/>` element.

```
<target name="run-my-xquery" depends="init">
  <property name="greeting.name" value="World"/>
```

```
<xquery outputproperty="xquery.result">
  <query><![CDATA[
    declare variable $addressee external;
    'Hello, ', $addressee]]></query>
  <param name="addressee" value="{greeting.name}"/>
  <database refid="MyDatabase">
    <library path="/MyLibrary"/>
  </database>
</xquery>
</target>
```

Example

This example loads the query from a file and stores the result in a file.

```
<target name="run-my-xquery-file" depends="init">
  <property name="greeting.name" value="World"/>
  <xquery outputfile="/tmp/xquery-out.txt">
    <query file="query.xq"/>
    <param name="addressee" value="{greeting.name}"/>
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </xquery>
</target>
```


A

- abstract schema, 126
- Abstract schema, 44
- admin client
 - creating database, 230
 - importing data
 - filter, 234
 - superuser password, 232–233
- Admin Client, 41
- administration client
 - adding indexes, 241
 - deserializing data, 238
 - deserializing users, 239
 - editing documents, 240
 - exporting data, 235
 - restoring data, 236
 - running queries, 241
 - serializing data, 237
 - serializing users, 239
 - starting, 228
- Ant
 - tasks, 317
 - types, 322
- Ant type, 318
- architecture, multi-node, 297
- ASmodel, 44
- attach, 274

B

- backend server, 297
- background server
 - starting on Unix, 70
- backup, 271
 - header, 266, 270
 - incremental, 263
 - info, 266, 270
 - restoring, 262
 - snapshot, 266
 - standalone, 263
- backup method
 - online, 268
- backup() method, 269
- backups, 262, 268, 270
- begin() method, 142
- binary large objects, 45
- <binding_server> element, 301
- BLOB
 - storing, 39
- bootstrap, 42, 66, 69
 - xhive-ant, 318
- bootstrap file, 69
 - multi-node configuration, 299
- branching, 41
 - methods, 124

C

- catalog, 221, 224
 - adding models, 222
- change identity
 - non-primary node, 314
 - primary node, 315
- checkpoint, 87
- checkpoint() method, 142
- client, 38
- Client.lax file, 68
- cold backup, 262, 268
- command
 - xdb admin, 62
 - xdb backup, 62, 263
 - xdb backup-library, 268
 - xdb configure-federation, 62
 - xdb create-database, 62
 - xdb create-federation, 62, 250
 - xdb delete-database, 62
 - xdb info, 62
 - xdb restore, 62, 264
 - xdb run-server, 62
 - xdb run-server on Unix, 70
 - xdb stop-server, 62
 - xdb suspend-diskwrites, 62
- command-line client
 - interactive console, 247
 - options, 247
 - xdb command, 246
- command-line client commands, 248
- command-line settings, 66
- commit() method, 142
- concurrent index, 167–168
- configuration file, 46, 230
 - binding_server, 301
 - file, 232
 - node, 301
 - segment, 231
 - xhive-clustering, 231
- configuration files, 68
- connect() method, 142
- consistency check, 283–284
- context conditioned index, 168
 - creating, 169
- context parsing, 100
- createDatabase() method, 84
- createSession() method, 141

D

- data
 - deserializing , 238
 - exporting, 235
 - importing, 234
 - restoring, 236
 - restoring from log file, 265
 - serializing, 237
- database, 42

- BLOBs, 45
 - catalogs, 44
 - checking consistency, 284
 - configuration file, 46
 - connecting, 85, 271
 - creating, 42, 62
 - documents, 44
 - files, 46
 - groups, 44
 - indexes, 44
 - libraries, 44
 - referencing objects, 144
 - segments, 45
 - superuser, 42
 - users, 44
 - database configuration, 86
 - dedicated page server, 70
 - dedicated page server program, 92
 - detach, 274
 - detach point, 47
 - detachable
 - library, 274
 - detachable library, 272
 - unusable, 275
 - disconnect() method, 143
 - distributed deadlock, 305
 - document
 - branching, 41
 - creating, 108
 - exporting, 120, 239, 272
 - linking, 40
 - normalizing, 102
 - parsing, 99
 - parsing with context, 100
 - publishing
 - PDF, 121
 - using XSLT, 121
 - retrieving, 109
 - by ID, 111
 - by library path, 112
 - by name, 111
 - previous versions, 124
 - using indexes, 112
 - using XQuery, 112
 - storing, 103
 - traversing, 116
 - using DOM, 116
 - using function objects, 119
 - validating, 101, 224
 - XQuery access, 178
 - documents
 - editing, 240
 - DOM
 - configuration, 103
 - retrieving documents, 110
 - support, 39
 - DTD
 - managing, 71
 - troubleshooting, 72
- ## E
- element name index, 167
 - element name index example, 167
 - exporting
 - documents, 239, 272
 - libraries, 239, 272
 - external editors, 71
- ## F
- failover, 294
 - federation, 42
 - creating, 250
 - creating replica, 290, 293
 - log, 43
 - read-only, 281
 - replicating metadata, 291
 - sets, 281
 - creating, 282
 - using, 92, 282
 - file system performance, 77
 - files, 46
 - fips, 67
 - FTP, 71
 - full-text index, 163
 - create, 164
 - full-text queries, 195
 - anyall options, 197
 - Boolean queries, 202
 - cardinality option, 198
 - limitations, 195
 - positional filters, 197
 - score calculation, 199
 - score variables, 198
 - thesaurus, 196
 - thesaurus handler, 196
 - wildcards, 195
 - xhive:fts function, 201
 - functions
 - external, 177
- ## G
- group
 - managing, 47, 90
- ## H
- high-availability, 314
 - host replacement, 314
 - hot backup, 262, 268

I

- ID attribute index, 166
- id collision, 274
- incremental backup, 262–263, 268
- index, 153–154, 160–161
 - adding, 241
 - concurrent, 167–168
 - context conditioned, 168
 - element name, 167
 - full-text, 163–164
 - ID attribute, 166
 - ignoring, 170
 - library, 165
 - library ID, 165
 - library name, 165
 - live, 150
 - metadata full text, 164
 - metadata value, 164
 - optimizing performance, 170
 - path, 151
 - query performance, 150
 - scope, 170
 - selectivity, 170
 - types, 150
 - using with XQuery, 188
 - value, 161–162

J

- JAAS, 97
- Java, 49
- Java command line
 - classpath, 50
- JDK, 49
- join() method, 143

K

- keep-log-files, 43, 263

L

- lazy replication, 289
- leave() method, 143
- library
 - backing up, 267
 - using API, 269
 - creating, 88
 - detach point, 46
 - detachable, 46, 272
 - exporting, 239, 272
 - ID index, 165
 - index, 165
 - metadata, 126
 - name index, 165
 - restoring
 - using API, 269
 - root, 44
 - unusable, 275
 - XQuery access, 178
- locking
 - context, 133
 - namebase, 134
- locking context, 133
- locking rules
 - multi-node configuration, 305
- log, 42
- log files, 43
- lucene, 67
- lucene blobs, 153

M

- master, 289
- memory, 66
- message logging, 287
 - java.util.logging, 286
- message logging areas, 287
- metadata
 - backup, 266, 270
 - indexing, 191
 - replicating, 291
- metadata full text index, 164
- metadata value index, 164
- model
 - adding, 222
 - linking, 222
- models, 44
- monitoring
 - statistics, 277–278
- move, 274
- multi-node architecture, 297
- multi-node configuration
 - API examples, 310
 - applications, 306
 - bootstrap file, 299
 - node server, 297
 - primary server, 297
 - upgrade, 301
- multi-node locking rules, 305
- multipath index, 153–154, 160–161
 - Lucene segment, 153
 - merge, 156
 - specification, 155
 - sub-index, 153
- Multipath Index Limitations, 160
- multipath index merge
 - performance, 156

N

- namebase, 134
- naming convention, 274
- node identity
 - changing, 314
- node identity change, 314
- node versioning, 125
- non-XML data, 106

O

- offline backup, 262, 268
- online backup, 262, 268
- online backup method, 268
- OSGi, 96

P

- page cache, 57
- page server, 38, 297
 - configuring, 70
- page server port, 56
- Page server settings, 67
- parallel queries, 215
- parsing, 223
- path index, 151
 - specification, 152
- performance
 - cachepages, 75
 - configuring JVM and
 - cache pages, 75
 - disabling disk-write caches, 78
 - file system, 77
 - internal server, 75
 - multiple disks, 77
 - page size, 77
 - parallel queries, 215
 - using indexes, 192
 - XQuery tuning, 204
- primary copy replication, 289
- primary server, replacing, 315
- property
 - xhive.bootstrap, 69
- PSVI, 194, 224
- PSVI information, 225

Q

- queries
 - running, 241
- query
 - preparing, 217
- queryable, 125
- quick start, 35

R

- RAM segment, 281
- range queries, 191
- read-only federations, 281
- read-only transactions, 148
- rename, 274
- replica
 - creating, 290, 293
- replication
 - lazy primary copy replication, 289
 - moving master, 292
 - removing replica, 292
 - running replicator, 291
 - using as failover, 294
- REST API, 246
- restore() method, 269
- restoring
 - from log file, 265
- restoring backups, 262
- rollback() method, 142
- RPC tracing, 78
 - console, 80
 - file, 80
 - session level, 81
 - system level, 80

S

- sample
 - running, 84
- scope, 86
- score customization, 161
- search
 - versions, 125
- segment
 - temporary, 46
- segments, 45
- serialization, 239, 272
- server, 38, 56
- Server.lax file, 68
- session, 86
- sessions, 133–134
 - connect() method, 142
 - createSession() method, 141
 - disconnect() method, 143
 - join() method, 143
 - joined, 136
 - leave() method, 143
 - lifecycle, 134
 - locking conflicts, 146
 - pools, 136
 - terminate() method, 143
 - transaction isolation, 146
- slave, 289
- snapshot backup, 262, 266, 268
- SSL, 283
- standalone backup, 262, 268
- statistics, 277–278
- superuser, 42

T

- temporary data, 46
- terminate() method, 143
- terms, 201
- trace file properties, 80
- tracing
 - RPC, 78
- transaction log, 289
- transaction recovery
 - multi-node, 299
- transactions, 86, 134
 - begin() method, 142
 - checkpoint() method, 142
 - commit() method, 142
 - distributed deadlock, 305
 - locking, 133
 - namebase and locking, 134
 - read-only, 148
 - rollback() method, 142

U

Unix
 background server, 70
 unusable detachable library, 275
 upgrade
 multi-node configuration, 301
 user
 managing, 47, 90
 users
 deserializing , 239
 serializing, 239

V

validated parsing, 223
 value index, 161–162
 type, 161
 versioned document, 123
 versioning, 123
 node, 125
 versions
 search, 125

W

web client, 246
 Windows service, 56

X

xDB
 commands, 61
 features, 37
 installing on UNIX, 59
 installing on Windows, 50
 uninstalling, 61
 xdb admin command, 62, 228
 xdb backup command, 62, 263
 xdb backup-library command, 267–268
 xdb command
 syntax, 247
 xdb configure-federation command, 62
 xdb create-database command, 62, 251
 xdb create-federation command, 62, 250
 xdb delete-database command, 62
 xdb info command, 62, 148, 251
 xdb restore command, 62, 264
 xdb restore-library command, 267
 xdb run-server command, 62
 xdb run-server command on Unix, 70
 xdb stop-server command, 62
 xdb suspend-diskwrites command, 62
 xdb.properties file, 66
 xhive-ant
 run java samples, 84
 xhive.bootstrap property, 69

xhive-ant, 318
 xhive:fts function extends XQuery, 201
 XHIVE_HOME, 67
 XhiveGroupIf, 91
 XhiveGroupIf interface, 91
 XhiveGroupListIf, 91
 XhiveGroupListIf interface, 91
 XhiveUserIf interface, 90
 XhiveUserListIf interface, 90
 XLink, 40, 122
 XQuery, 39
 accessing documents and libraries, 178
 collation support, 208
 collection(), 178
 data model, 213
 doc(), 178
 error reporting, 178
 extending using Java, 218
 extension expressions, 179
 extension function xhive:highlight, 186, 188
 extension functions, 182
 external variables, 175
 full-text queries, 194
 full-text support, 208
 implementation, 199
 instance methods, 218
 Java objects, 218
 limitations, 219
 methods, 173
 modules, 209
 multiple indexes, 191
 name element index, 189
 namespace declarations, 213
 options, 179
 parallel queries, 215
 preparing queries, 217
 proprietary extensions, 193
 range queries, 191
 security, 209
 security policy, 209
 supported, 207
 type checking, 219
 unsupported, 207
 updates, 211
 using, 173
 using indexes, 188
 using type information, 194
 using type information sample, 215
 value index, 189
 XML Schema, 210
 XQuery collation support
 java, 206