# EMC® Documentum® xDB

## Version 10.0

## Manual

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

# Table of Contents

# Preface

## xDB Manual

This manual is designed to provide a technical introduction to the xDB product. This manual describes all aspects of installing, configuring, programming with, administering, and using xDB.

This manual contains the basic xDB information. For more information about specific aspects of xDB and how to use xDB in combination with other tools, see the EMC Developer Network.

## Intended audience

This manual is for xDB users and developers. This manual assumes that the reader is familiar with Java, XML, operating systems, basic database principles such as transactions, locking, and access rights.

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

## Organization

This manual is structured to provide basic and essential information first, then moves on to more advanced topics.

## Typographic conventions

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

**Table 1 Typographic conventions**

| Appearance | Meaning |
|---|---|
| *Fixed terms* | 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 xDB's Java API. |
| 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 xDB installation directory. |
| *Variable Names* | Variables in command strings and user input variables |

# Resources

The Documentum Developer site has resources related to xDB, along with case studies and sample code that you can download. For more information, see EMC Developer Network.

# Support information

EMC Documentum technical support services are designed to make deployment and management of Documentum products as effective as possible. The *Customer Guide to EMC Software Support Services* provides a thorough explanation of the support services and policies. This document is available on the Powerlink website (http://powerlink.emc.com).

# Revision History

## *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 namebases are now concurrent. The options XhiveLibraryIf.CONCURRENT_NAMEBASE 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 9.0.5

- 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 XhivePrepardeQueryIf#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 unziped.
- 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.

## *Changes in release 8.2.1*

- Add new LSParser and LSSerializer option "xhive-raw-attributes" to store unparsed values of attributes (with entity/character references preserved) and use them during serialization.
- Add new API XhiveFtsUtilIf.validateQuery to validate the syntax of a full text query without executing it.
- Fix problem with XQuery node constructor copying an entity reference node with a character reference created by parsing with the "xhive-character-references" option.
- Fixed a bug in the matching of Phrase queries using wildcards in one of the terms.
- Deleting a concurrent path index with composite keys could leave behind a few bytes of garbage.

## Changes in release 8.2.0

- Add new LSParser options "xhive-predefined-entities" and "xhive-character-references" to store predefined entities and character references as entity reference nodes in the resulting document.

## Changes in release 8.1.3

- Fix several problems in the XQuery group-by optimization.
- Exporting an empty blob from the admin client would hang.
- Fix XQuery parse error on empty attribute value.
- DOM methods Element.getAttribute(Node)NS and setAttributeNS now correctly treat an empty namespace URI as null.
- XQuery functions xhive:document-name, fn:document-uri and fn:node-kind now work on blobs.
- Changed admin client lock wait time to 5 seconds.
- Fix path processing error at the <upload/> Ant task.
- We now throw a real org.w3c.dom.DOMException on DOM errors, instead of a subclass with an obfuscated name.
- If a deadlocked transaction was not rolled back, a later deadlock involving the same transaction could go undetected.

## Changes in release 8.1.2

- Fix rare case where log files could remain in the log directory indefinitely after a crash.
- Fix getCreated(), getLastModified() and getOwner() for indexes. Note that concurrent indexes use the values of their owner.

## Changes in release 8.1.1

- Avoid a case where all free space would be reserved for a single session, causing other concurrent sessions to allocate space by extending a data file unnecessarily.
- Fix rare erroneous OBJECT_CLOSED exception when removing the version space of a document or blob.
- An XQuery element constructor containing a document node with a document type would attempt to append the document type node to the element and fail. The document type node is now skipped.
- Processing instructions in DTDs are now preserved.
- Ant tasks <valueindex/> and <metadatavalueindex/> allow setting the type of the indexed value to all supported types, and not just "String".

## Changes in release 8.1.0

- Add new API XhiveXQueryQueryIf.setDocumentFilter() to filter the documents seen by the query.
- XQuery extension functions can now return Iterators to allow incremental generation of result sets.
- Indexes can now be compressed to take less disk space. (Does not apply to concurrent indexes.)
- Add admin client feature to reset database administrator password by superuser.
- Added new Ant task <upload/>.

## Changes in release 8.0.2

- Fix possible ArrayIndexOutOfBoundsException in new segment layout code when releasing pages.
- The LSParser DOMConfiguration option "xhive-store-schema-only-internal-subset" now not only avoids storing the external DTD subset, but also any XML Schema.
- Fix bug where revalidation of documents agains an XML Schema with local element declarations and elementFormDefault=’unqualified’ failed.
- The ELEMENT_CHILD_NUMBER_ERROR exception now gives the name of the index in the message.

## Changes in release 8.0.1

- Fix rare bug in recovery of concurrent indexes that could corrupt the index and cause federation recovery to fail.
- Add new API XhiveDriverIf.setConnectionTimeout() to set timeout on connection from client to server.
- Avoid hanging when listener thread accept fails due to IO errors like "too many open files".
- Reduce some usage of memory allocated until transaction end during some DOM operations.
- Parsing a document with an explicit schema location set on the parser, but with "xhive-store-schema" set to false, used to take an unnecessary write lock on the catalog.
- Added new Ant task <closedriver/>.
- Corrected omission in Javadoc of XhiveListIndexIf.addMetadataValueIndex. These indexes *can* use type information.

## Changes in release 8.0.0

- New databases created with X-Hive/DB 8 (and new segments in old databases) use a new internal layout that should reduce fragmentation of large objects. This should speed up reading large blobs and documents.

- The cache page replacement algorithm has been rewritten to be more resistant to reading large infrequently used data, like large blobs.
- When using the database in client/server mode, full text search phrase and boolean query processing is now done at the server to reduce the necessary communication.
- Implemented metadata index lookup omission at clauses with intersections between metadata indexes. Using heuristics (currently the ratio of nodes per key in the index) to determine which index will be selected for use. See also the documentation of the option "xhive:ignore-indexes".
- Many interfaces now extend the new XhiveObjectIf that allows retrieving the session that an object belongs to.
- Many interfaces in com.xhive.dom use Java 5 covariant return types to return more specific types for DOM methods. This should reduce the need for casting to the X-Hive/DB interfaces.
- Created new interface com.xhive.util.interfaces.IterableIterator<T> to allow using Java 5 style foreach loops with methods returning Iterators.
- Added a way to wait for a child to be added to concurrent library, see XhiveLibraryIf.asBlockingQueue().
- The XhiveIndexIf and XhiveLibraryChildIf interfaces have a new method getStoragePages to retrieve the number of disk pages that they use.
- Add new API XhiveLibraryChildIf.unlockFromParent() to clear the lockWithParent flag from a library child.
- The server can now store stack traces where locks are acquired to facilitate debugging lock related problems. See the new method XhiveDriverIf.setStoreStackTraceInLock.
- The name of a session can now be set after it was created using XhiveSessionIf.setSessionName(String name).
- There are new methods XhiveSessionIf.interrupted() and XhiveSessionIf.isInterrupted() to check the interrupt status.
- XhiveSessionIf.getUser().getName() no longer needs to read lock the user/group storage.
- In client/server mode, more requests will retry to establish the connection to a restarted server.
- It is now possible to remove the last version of a branch as well as older versions with XhiveVersionIf.remove().
- Allow metadata of versions to be set through the setValue() method of the entries in the entrySet of the metadata.
- Add interfaces XhiveMetadataMapIf and XhiveMetadataEntryIf to allow finding out whether metadata entries are versionable.
- Metadata of old versions can now be changed using the map returned by XhiveVersionIf.getMetadata().
- Implement the current W3C XQuery Update Syntax working draft. The implementation should conform to the version from 2007-08-28, available at http://www.w3.org/TR/2007/WD-xquery-update-10-20070828/. As this recommendation is in a very early stage, substantial changes to the standard and our own implementation have to be expected.
- Implemented the W3C XQuery Update statement placement rules. Update statements (update expressions like "do delete" and functions that perform updates) are now only allowed on the main execution branch of the Query, see the manual for more details.
- Calling code can now restrict what executed queries can do, in case the queries are entered by untrusted users. See new interface com.xhive.query.interfaces.XhiveXQueryPolicyIf.
- Added a new Java extension mechanism to XQuery. Java modules can now be implemented as plain Java code and imported by specifying the fully qualified class name. E.g. `import module namespace my = "java:com.example.my.XQueryLibrary"; my:foo(), $my:bar`
  See the manual for more details.
- fn:collection() now also works without any parameter, it defaults to the library where the XQuery was created.
- The implicit timezone (fn:implicit-timezone) is now set by default to the local timezone, and not to GMT.

- Parser error messages now give the actual token (";") instead of the identifier (SEMICOLON).
- Give column numbers in error messages where possible.
- Validate expressions should now give more verbose error messages on failure
- Automatic type conversion from Java variables, e.g. on the result of an xhive:java() call, on external function results or on external variable results, will now import org.w3c.dom.Nodes from other DOM implementations.
- Full text phrase searches can now use terms with wildcards, e.g. "foo bar*".
- Implemented efficient leading wildcard searches (i.e. "*pattern") on full text indexes. See the option "FTI_LEADING_WILDCARD_SEARCH" in the "Full Text Indexes" section of the manual.
- At a full text search, you can now specify which is the implicit conjunction to be used, either using xhive:fts-implicit-conjunction or Settings.XHIVE_FTS_IMPLICIT_CONJUNCTION. The only valid values are "AND" and "OR"
- Implemented support for full-text fields in path value indexes. See the section "Path indexes" in the manual.
- The X-Hive/DB ant tasks (previously a separately downloadable package) have now been integrated with the core distribution. See the chapter "Ant tasks" in the manual.
- New ant tasks added: <listindexes/>, <renamedatabase/>, <copydatabase/>, <registerreplicator/>, <unregisterreplicator/>, <replicatefederation/>.
- Ant index adder tasks now support a 'unique' attribute, default is false.
- Ant task <backup/> and the command line tool XHBackup both got a "overwrite" option, to determine whether or not to overwrite an existing destination file. Default is false. Previously both tools would overwrite an existing file.
- The replaceChild() method used on a concurrent library no longer takes an unnecessary read lock on the next sibling of the replaced library child.
- If the server has no memory to create a thread for a client session, it will disconnect that session and log an error instead of stopping the server completely.
- Avoid case where internal thread could have an open transaction that caused driver.close() to fail with TRANSACTION_STILL_ACTIVE exception.
- Per the latest working draft, the XPointer function "range" has been renamed to "covering-range".
- Many fixes to improve compliance to the XQuery 1.0 Recommendation that has now been released. Some of these are listed explicitly below.
- Functions in the XQuery namespace are no longer allowed. Functions declared in XQuery are in the XQuery function namespace by default, e.g. `declare function foo() external;` gives a function {http://www.w3.org/2005/xpath-functions}foo/0. This is illegal according to the W3C XQuery specification. Users are required to put functions either in a custom namespace, or use the predefined local namespace. `declare namespace ex = 'http://www.example.com';`
```
(: ok - in example namespace :)
declare function ex:foo() { 123 };
(: ok - in predefined XQuery local namespace :)
declare function local:foo() { 123 };
```
Of course these namespaces have to be used in function calls, too.
- Supporting the above, XhiveXQueryQueryIf.setFunction(localname, function) now puts the function in the local namespace instead of the XPath functions namespace.
- Functions with an empty namespace URI are not permitted any more. E.g. `declare default function namespace = "";`
```
declare function foo() { "bar" };
```
will give an error now.
- Functions are now resolved statically in XQuery. The only user visible change is that it was formerly possible to use an external function without declaring it, now declaring is always required.
- Completely overhauled the parser for date-, time- and duration values. This fixes several corner cases where values were accepted even though they did not exactly meet the specification.

- Moved the implicit-timezone option to the xhive: namespace, e.g. `declare option xhive:implicit-timezone "PT10H";`
  (options have to be in namespaces)
- fn:deep-equal() now properly checks its parameters.
- fn:idref() now also works for documents that use DTDs
- XML elements of type xs:hexBinary and xs:base64Binary are now correctly converted to typed XQuery values.
- Compare operations on values now also report the operator if the values were uncomparable. Please note that many XQuery types support '=' and '!=', but not '>', '<' and the like.
- Fixed a regression bug where undefined external functions would result in NullPointerExceptions.
- Fixed a bug in the handling of fn:replace()'s replacement string. Referring to a capturing group that does not exist will no longer cause an error: `(: $2 does not exist, is treated as empty string :)`
  `fn:replace("foo", "(o)o", "$2a") ==> "fa"`
  also, additional digits will no longer be swalloed `(: $12 does not exist, so this is interpreted as $1 :)`
  `fn:replace("foo", "(o)o", "$12a") ==> "fo2a"`

- The "declare copy-namespaces" option is now recognized, even though still unsupported. Use XhiveNodeIf.normalize() and XhiveDocumentIf.normalizeDocument() with the namespaces option set in the DOMConfiguration to get a namespace fixup.
- The XQuery Lexer now requires non-delimiting terminals to be surrounded by either whitespace or delimiting terminals as the spec requires. This means the Lexer will give an error on code like: `10 div3 (: error!, use 10 div 3 :)`

- Casting to an xs:QName is now only allowed if the source of the case is a string literal, as required by the specification. E.g. `"foo" cast as xs:QName (: legal :)`
  `let $bar := "bar"`
  `return $bar cast as xs:QName (: error! :)`
  Use fn:QName when needed.
- Castable and cast expressions no longer allow xs:NOTATION as a target type (xs:NOTATION is defined to be an abstract type).
- Fix some corner cases for comparison expressions, X-Hive should now exactly implement the table at http://www.w3.org/TR/xquery/#mapping This means in particular that xs:dateTime values are not comparable using 'gt', 'lt' etc., only 'eq' and 'ne' (and the corresponding general comparisons) work.
- Fixed fn:string-join() function signature, the second parameter is now always required.
- fn:sum() and fn:avg() now check the type of it's parameter correctly for single values. They now also work on xs:dayTimeDuration and xs:yearMonthDuration. They now correctly return an xs:double if called with a single xs:untypedAtomic parameter.
- fn:min() and fn:max() now correctly check the type of their parameters for single values.
- Adjusted xhive:if-absent() and xhive:if-empty() to the specification, they both return atomized values now.
- The regular expression implementation backing fn:replace() and fn:matches() now supports character group substraction, e.g. [m-p-[no]] matches 'm' and 'p'.
- Moved the "fn" and "local" prefixes to the updated namespace URIs, fn now binds to "http://www.w3.org/2005/xpath-functions" and local binds to "http://www.w3.org/2005/xquery-local-functions".
- Removed the obsolete predeclared namespace 'xdt' from XQuery.
- CDATA sections are now only allowed within direct element constructors, e.g.
  `<foo><![CDATA[<!>]]></foo> (: legal :)`
  `<![CDATA[<!>]]> (: error! :)`

- XML Schema types in "treat as" and "instance of" expressions are now checked statically and will give errors for unknown types.
- Fixed a lexing bug with comments close to itemtype declarations; allow comments in even more places.
- Fixed a parser bug with unary arithmetic expressions containing multiple consecutive operators (e.g. ++-+–3 gave a parse error).
- Fixed a bug where XQuery modules were allowed to contain garbage at the end of the module.
- Whitespace within xs:base64Binary values is no longer an error.
- Fixed a bug where updates to nodes covered by path value indexes could cause additional nodes to be inadvertently removed from the index.
- Fix bug with full text phrase queries using metadata full text indexes that could give incorrect results or exceptions.
- The included version of Xerces has been upgraded to 2.9.1.
- The included version of ICU4J has been upgraded to 3.8, which includes support for Unicode 5.0.
- The included version of FOP has been upgraded to 0.94 and the XhiveFormatterIf interface has been updated.
- The included version of Lucene has been upgraded to 2.2.0. Note that it no longer defines ’s’ and ’t’ as English stop words.
- The included ant.jar has been upgraded to version 1.7.0.

# Chapter 1

# Quick Start

This chapter contains the following topics:



## Getting started with xDB

This quick start section provides a high-level introduction to help users get started with xDB. It describes all the necessary steps to get xDB running, from installing xDB, to creating a database, and running a first sample. For more detailed information about installing xDB, see Installing xDB, page 33.

## Pre-installation requirements

Before installing xDB, verify the following:

- When installing on Windows 2000, you must be logged on as an administrator or a user with similar access privileges.
- The system on which you are installing xDB has the Sun JDK 6 or a fully compatible Java Virtual Machine (JVM) installed. Install the JDK before installing xDB.

## Installing xDB

If you have previous versions of xDB running, you must uninstall them first or choose other directories and port-numbers.

To install xDB:

- On Windows, run the **setup.exe** file and follow the instructions on the screen.
- On UNIX, extract the distribution and run **sh setup.sh**.

  The UNIX installation requires some post-installation steps, which are described in the readme.txt file that is part of the distribution. You can install the xDB program under any account.

### Related links

Detailed installation instructions for Windows

Detailed installation instructions for UNIX

# Creating a database

Using the administrator client is the easiest way to create a database. Alternatively, the xdb create-database command can be used.

To create a database using the administrator client:

1. Start the administrator client by running the **xdb admin** command, which is located in the bin subdirectory of the xDB target directory and in the Start menu group of xDB on Windows.
2. Select the menu option **Database** > **Create database** to create a database.
3. Create a database named **united_nations** with the superuser password as entered during installation of xDB and administrator password **northsea**. By default, this database is used in all samples described in this manual. The other fields should be left unchanged.

   After creating the database, you can close the administrator client.

# Running a sample

xDB samples are run using the Ant build system. The **xhive-ant** command sets the proper CLASSPATH and other parameters. All sample source is located in the *XhiveDir*\src\samples\manual directory. Before running the samples, verify that the property values in the SampleProperties.java file match your settings.

To run a sample:

1. Open a command prompt and navigate to the ***XhiveDir*\bin** directory.
2. Run a sample that inserts two documents into the database using the **xhive-ant** command:

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

# Chapter 2

# xDB Concepts and Architecture

This chapter contains the following topics:



## xDB concepts

xDB is a database that enables high-speed storage and can handle an extensive amount of XML documents. Using xDB, programmers can build custom XML content management solutions that are fully tailored to the exact requirements of any given application. xDB stores XML documents in an integrated, highly scalable, high-performance, object-oriented database. xDB exposes the database and its contents via an application programming interface (API). The API is written in Java.

xDB implements and extends the following recommendations of the *World Wide Web Consortium (W3C)* for querying, retrieving, manipulating, and writing XML documents:

- Document Object Model (DOM) Level 1
- DOM Level 2 (Core and Traversal)
- DOM Level 3 (Core and Load/Save)
- The eXtensible Stylesheet Language - Transformation (XSLT)
- XQuery
- XPath
- XLink
- XPointer

### *DOM support and data manipulation*

xDB implements an extended DOM level 3 interface for manipulating content, structure, and style. xDB supports all DOM level 3 functionality, including functions for retrieval, modification and navigation within XML documents. DOM level 3 does not support XML collections for handling more than one document. Extended API functions in xDB provide support for processing multiple documents simultaneously.

xDB supports nested libraries. Libraries can be stored within other libraries. Libraries are implemented as DOM nodes, and all operations on documents, including XQuery queries, can also be performed on libraries.

# Searching, linking, and indexing

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. xDB includes an XQuery query engine implementation. For more information about using XQuery, see Using XQuery, page 159.

xDB supports XPath and XPointer queries, page 93.

XLink is a W3C recommendation that enables links between XML documents. XLink creates simple links equivalent to <a href> links in HTML, and 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 to the W3C XML Linking Language (XLink) Version 1.0 documentation.

Indexing enables faster access to parts of a library or document and increases the performance and scalability of xDB applications. xDB supports several different indexing methods:

- Context Conditioned Indexing , page 124
- Library ID and library name indexing , page 121
- Id attribute indexing , page 122
- Element name indexing , page 123
- Value indexing , page 118
- Full text indexing, page 120

# Session and transaction control

xDB includes a transaction mechanism to ensure 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 a session can be committed or rolled back.

### Related tasks

Creating sessions

Using transactions

# Administration

The xDB distribution includes an administration tool with a data browser that displays database content in a tree view. The administration tool provides access to user and group management features, such as configuring permissions for accessing content and running queries. The administration features are also accessible through the API.

### Related topics

Administration client

Command line client

Ant tasks

## Non-XML import and BLOB storage

xDB contains an SQL Loader that uses JDBC to import data stored in relational databases. An integrated version of the Xerces parser imports the XML documents. All JDBC-compliant relational databases are supported. Data in sequential files can be loaded into xDB using the SQL Loader interface.

### Related topics

Importing non-XML data

Storing BLOBs

## Versioning

xDB provides linear versioning with branches and can store multiple document versions within the context of the document. Storing multiple document versions allows users to track the changes in a document and restore older versions.

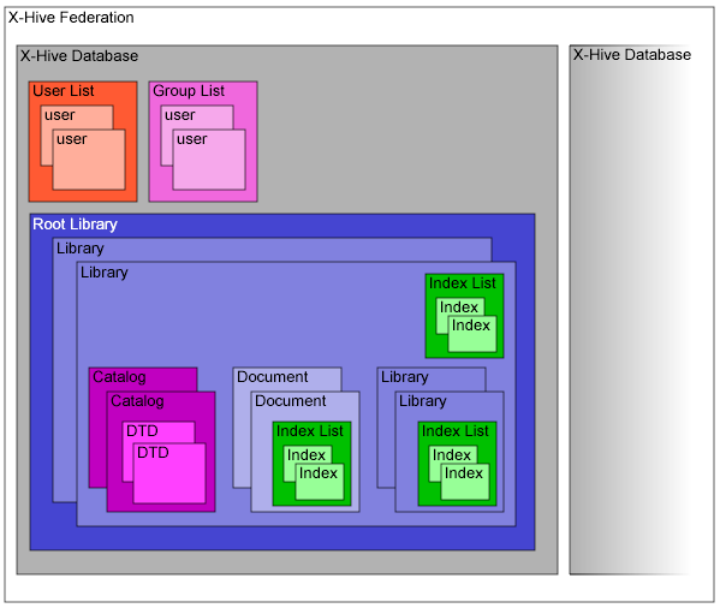
### Related topics

Using versioning

# xDB architecture

A single xDB application contains one or more databases. These databases are grouped within a single federated database.

**Figure 1 xDB federation**




A federated database contains the following objects:

## Databases

An xDB application can contain one or more databases. These databases are part of the federated database and can contain various objects, as described in xDB database objects.

**Table 2 xDB database objects**

| Database object | Description |
|---|---|
| Users, user lists | During the initial installation procedure, xDB creates a so-called superuser account for the database administrator. The administrator then creates user accounts for all other database users. Every xDB user has a database user account with a unique user name and password. Each account is assigned a set of permissions that specify the level of access the user has to the database.<br><br>xDB maintains a user list for each database. |
| Groups, group lists | A group is an object in the database that contains one or more users. The main purpose of a group is to assign the same access rights and privileges to a subset of users.<br><br>xDB maintains a group list for each database which contains all groups of that database. |
| Indexes, index lists | An index list provides a list of all the indexes of a library or document. |
| Libraries | A library stores documents or other libraries in an hierarchical structure.<br><br>The nested structure of libraries within xDB is very similar to the nested structure of directories or folders within a file system. Any library can contain other libraries and there is exactly one root library that is created automatically. |
| Documents | A document stores XML data. xDB can handle both valid and well-formed XML documents. |
| Catalogs, schemas | A catalog stores Document Type Definitions (DTDs) and XML schemas. |
| Binary large objects (BLOBs) | 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 next to XML documents allows managing all resources for a specific project or product in one uniform way. |

# Superuser

A federated database has one super user with the user name **superuser**. xDB creates the superuser during the installation process, but does not represent the superuser as object in the xDB API.

The super user has the permissions to create and delete databases, and to perform administrative operations. These actions include setting the license key and backing up the database. The super user cannot access regular data such as libraries and documents.

## Segments

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

By default, all data is stored in the default segment unless a user specifies a different segment. Data is never automatically stored in another segment, even if the current segment is full. A single segment with multiple files can be used to create an automatic overflow. The xDB API provides methods for assigning a segment for temporary data and for storing all children of a specific library.

## Database files

Each database segment contains one or more files and each file consists of one or more pages. Files can be added to the database using the administration client or the xDB API. By default, a segment always has at least one file, the default file. The default file can never be deleted without deleting its owner segment.

The maximum page size of a file is set when a federation is created. If a file exceeds maximum size, the overflow data is allocated to the next file of the segment. If all the files in the segment have reached their maximum size, any further allocation in the segment fails. The allocation is random and different pages of single document can be stored in different files. Using different segments is the only way to control where the data is stored.

The maximum page size can be set to unlimited. In this case, the next file of the segment is never used, not even if the disk is full. The only reliable way to detect a full disk is to write an empty page to the file while allocating the page.
**Note:** Sometimes xDB has to allocate pages for internal page allocation administration. In this case the maximum file size limit is ignored because the page allocation is mandatory to keep the data consistent. Therefore, a file can become slightly larger than the limit set by the administrator.

## Database configurations

A configuration file is used to specify an initial database configuration. The default database configuration has a single segment with a single file and all data clustered in the default segment. The administration client or the API can be used to change the default configuration, to create or delete segments and files, and modify default cluster rules.

The configuration of existing databases can be obtained in form of a DOM Document by using the XhiveDatabaseIf.getConfigurationFile() API method.

## Log files

xDB uses *Write Ahead Logging (WAL)* to ensure that data can be recovered. Data is first committed to the database log before being written to the actual database files. Log files are sequential and are written to the log directory that was specified when federation was created. If the keep-log-files option of the federation is turned off, log files that are no longer needed for recovery are automatically deleted. If the option is turned on, log files are only deleted after a backup.

## Detachable libraries

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 besides id and name indexes.

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. All the segments of the library must have the same state at any time. Once a library is detached, it is not accessible from the database.

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

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

## Database configuration files

Database configuration files are used to describe initial database configurations. They are only used when a database is created and do not affect existing databases. The configuration file is in XML format. The configuration file can have the following elements:

- <xhive-clustering>, page 30
- , page 31
- <file/>, page 31

The following example describes the configuration file for 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:

- , page 31

# 

The  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  element can have the following child elements:

- <file/>, page 31
- <binding_server/>, page 72

# <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. |

# Database consistency checker

The database consistency checker checks the physical and logical consistency of a database, such as the consistency of the internal database, segment administration, and data pages. Only the database administrator has the privileges to run this tool. The consistency checker can only be used when there are no active updating transactions, otherwise the result of the consistency checker is not defined.

There are different ways to run the consistency checker:

- The <checkdatabase/>, page 225 Ant task
- The <checklibrarychild/>, page 226 Ant task
- The <checkfederation/>, page 227 Ant task
- The <checknode/>, page 228 Ant task
- The XhiveConsistencyCheckerIf/XhiveFederationConsistencyCheckerIf API

## Related tasks

Checking database consistency

# Chapter 3

# Installing xDB

This chapter contains the following topics:



## Pre-installation requirements

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.

### Supported platforms

xDB is a Java application and can be installed on any platform that contains a Java Development Kit 6 or higher. Installers are available for Windows 2000/XP/Vista/Win7 and Unix and have been tested on Linux, Solaris, AIX, and Mac OS X).

**Table 5 Minimum system requirements for xDB**

| | |
|---|---|
| RAM | 256MB |
| Hard drive space for xDB software | 50MB |
| Hard drive space per database | 1MB |
| Java | JDK 6 or higher (http://java.sun.com) |
| Network | TCP/IP |

### Software requirements

xDB requires a Java Development Kit 6 or higher, or a fully compatible Java Virtual Machine. The xDB installation fails without JDK 6.

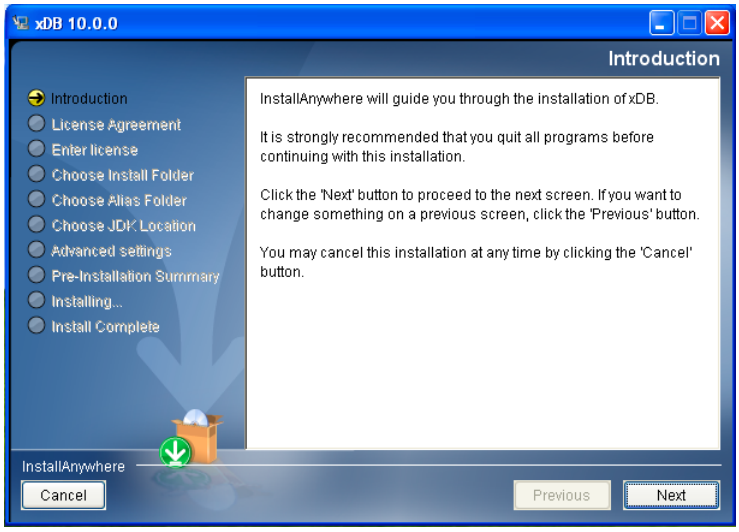
# Installing xDB on a Windows platform

The xdb_setup.exe program installs the xDB installation on a Windows machine. The program is located in the root directory of the xDB CD-ROM or distribution file. The installation program installs the files in the proper directories and augments the PATH environment variable.

To install xDB on a Windows platform:

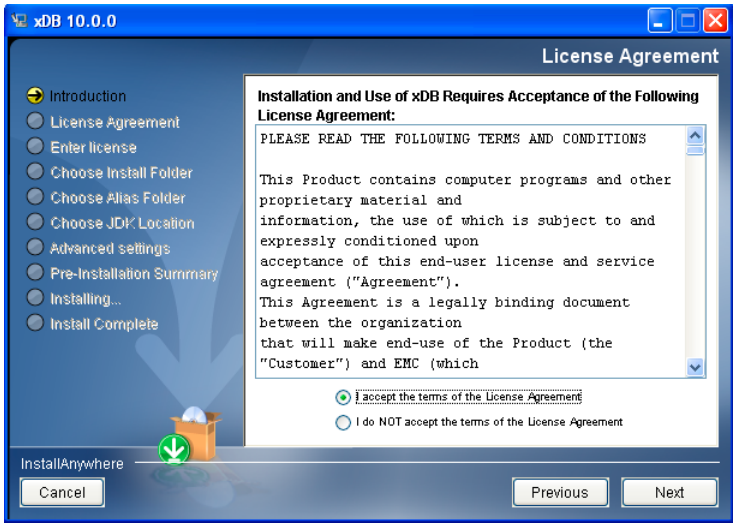
1. Double-click the xdb_setup.exe file.

   The xDB installation window displays.



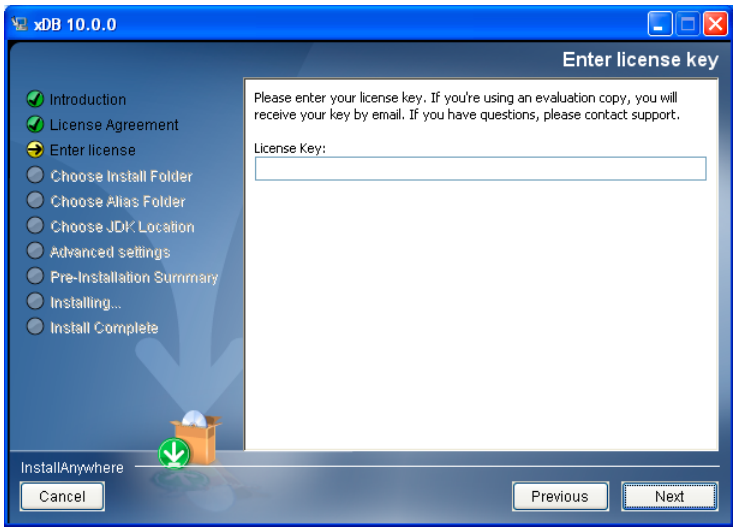

2. Click **Next**.

   The License Agreement window displays.




3. Accept the software license agreement, then click **Next**.

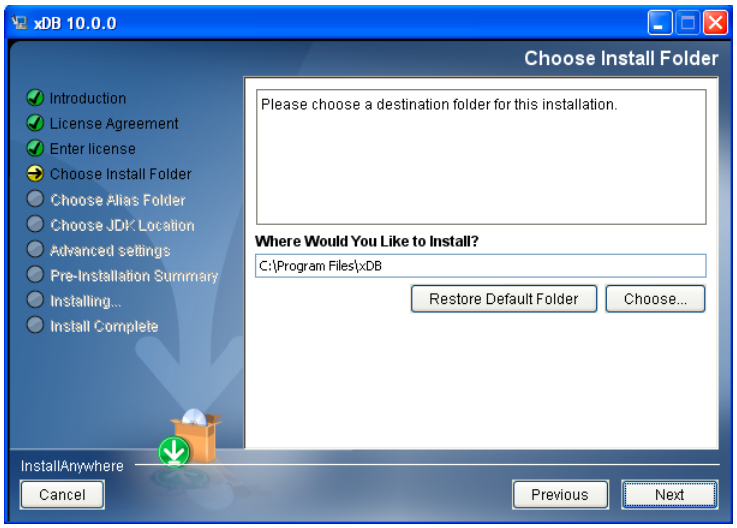
   The Enter license key window displays.




4. Enter the xDB license key you received, then click **Next**.

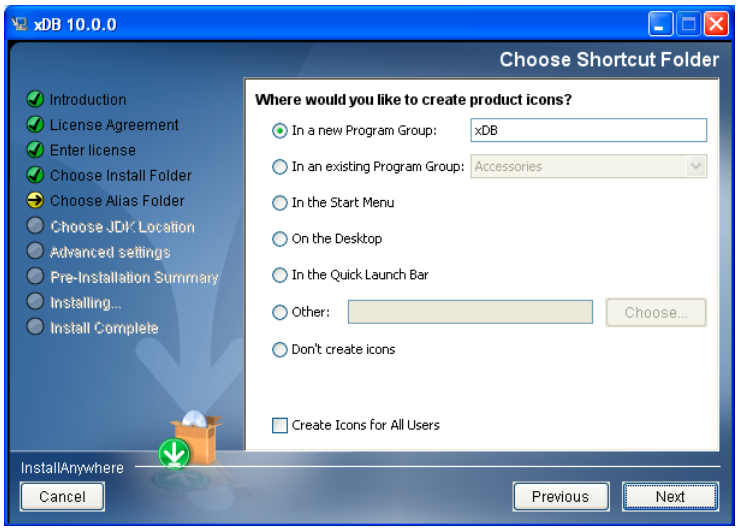
   **Note:** xDB software licenses are only valid for a limited time. If in doubt about your license, contact xDB customer support.

   The Choose Install Folder window displays.



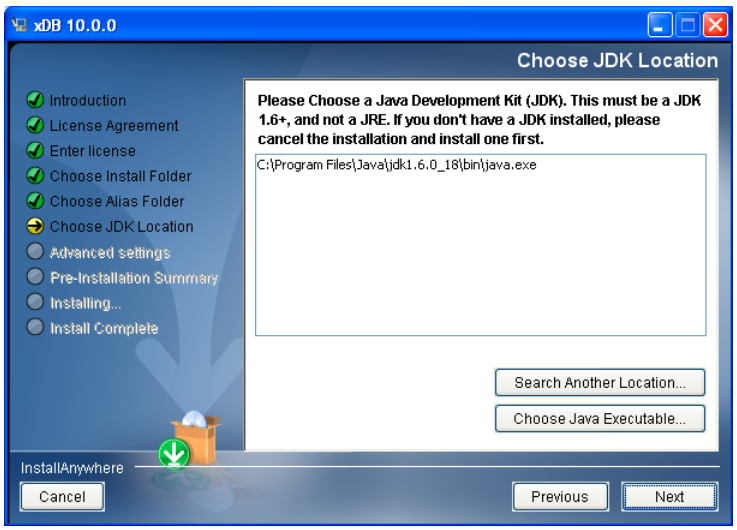

5. Specify the target installation directory, then click **Next**. The default directory in Microsoft Windows is C:\Program Files\xDB or similar, depending on the locale of your Windows installation.

   The Choose Shortcut Folder window displays.



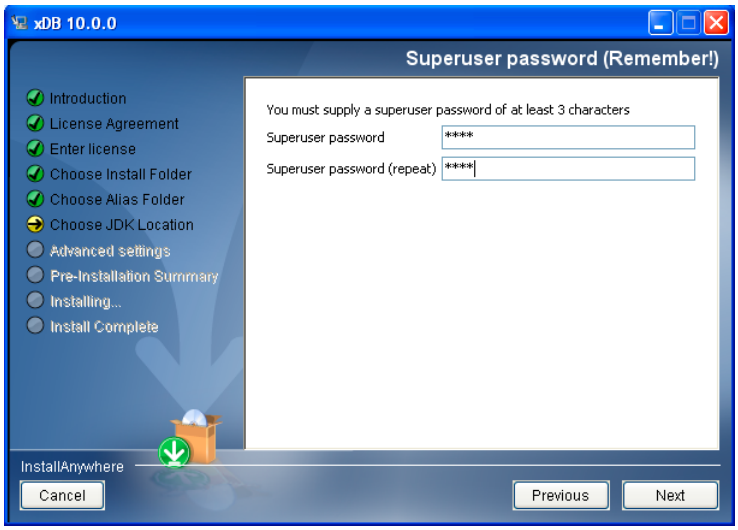

6. Select a place where shortcuts for xDB should be created, then click **Next**.

   The Choose JDK Location window displays.



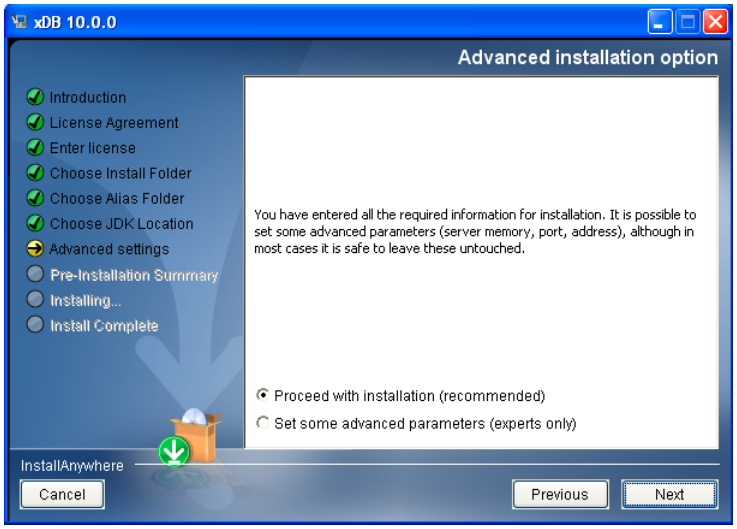

7. Click **Choose Java Executable** to browse for the applicable Java Development Kit for the path to the **java.exe** file, for example, C:\jdk1.6.0_07\bin\java.exe, then click **Next**.

   The Superuser password window displays.




8. Enter the password of the superuser then click **Next**. The superuser password is required for creating databases and other administrative tasks.

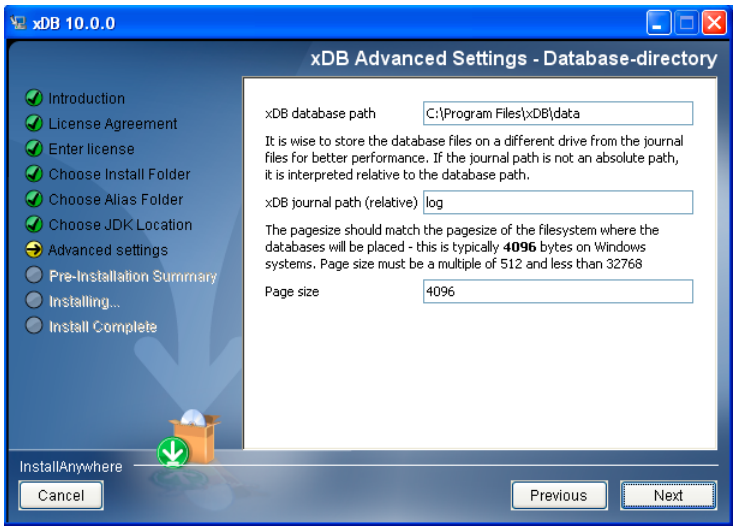
   The Advanced installation option window displays.




9. Select whether to proceed with a standard installation or whether to perform additional advanced installation steps, then click **Next**.

   If you selected **Proceed with installation**, the program files for xDB are now installed on your host machine. Proceed to step 9.

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

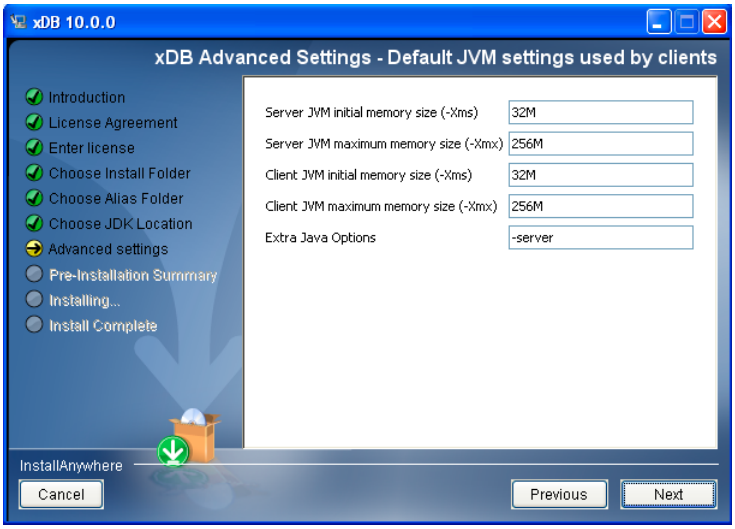



a. Specify the database path, xDB journal path, and page size, as follows:

| Advanced Setting | Description |
|---|---|
| xDB database path | The path to the database files. The default path is based on the installation directory. Log files should be stored on a different disk than database files for improved performance. |
| xDB journal path | The path to the database log files. Log files should be stored on a different disk than database files for improved performance. |
| Page size | The page size of files stored in the database. The page size should match the page size of the filesystem where the database shall reside. You should not pick larger sizes than the file system one. See page sizes, page 212 for more information. |

b. Click **Next**.

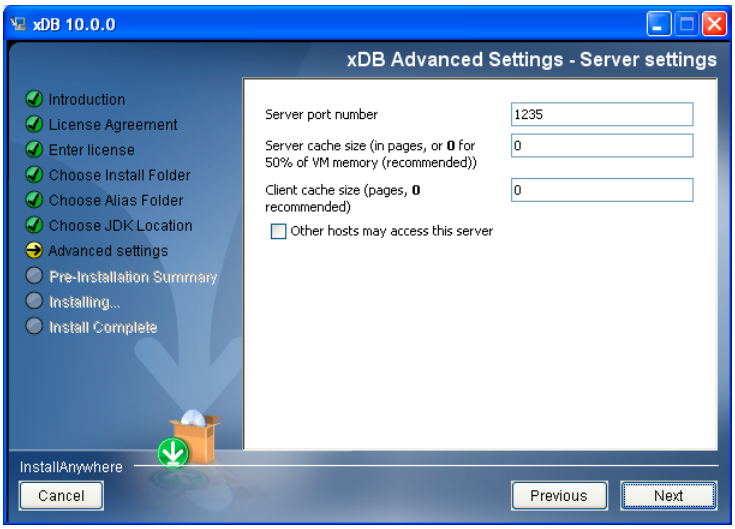
The xDB Advanced Settings - Default JVM settings window displays.




c. Set the JVM properties, as follows then click **Next**.

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

The xDB Advanced Settings - Server settings window displays.




d. Enter the server port number and server and client cache sizes.

xDB uses the server port number and cache size to install a service that acts as a page server, as follows:

| Server settings | Description |
|---|---|
| Server port number | The port number of the page server. The server can run on any port. xDB checks whether the port is available. The port-number determines the URL which is used to access the database. |
| Server cache size | The cache size determines the amount of pages cached by the server to speed up performance. We recommend setting the cache size to a default size of 0. If the cache size is set to 0, the database uses half of the JVM memory as cache. |

| Server settings | Description |
|---|---|
| Client cache size | Clients connecting to the server, such as the Admin client or the command line, 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 have processes on multiple machines accessing the database. By default, only processes running on the same machine can access the server for security reasons. |

e. Click **Next** to finish the installation.

If xDB installed successfully, the installation program displays a message confirming that the installation completed without errors.

# Installing xDB on a UNIX platform

On UNIX platforms, xDB can be installed as any user.

To install xDB on a UNIX platform:

1. Extract the distribution **.tar.gz** file to the directory where you would like to install xDB.

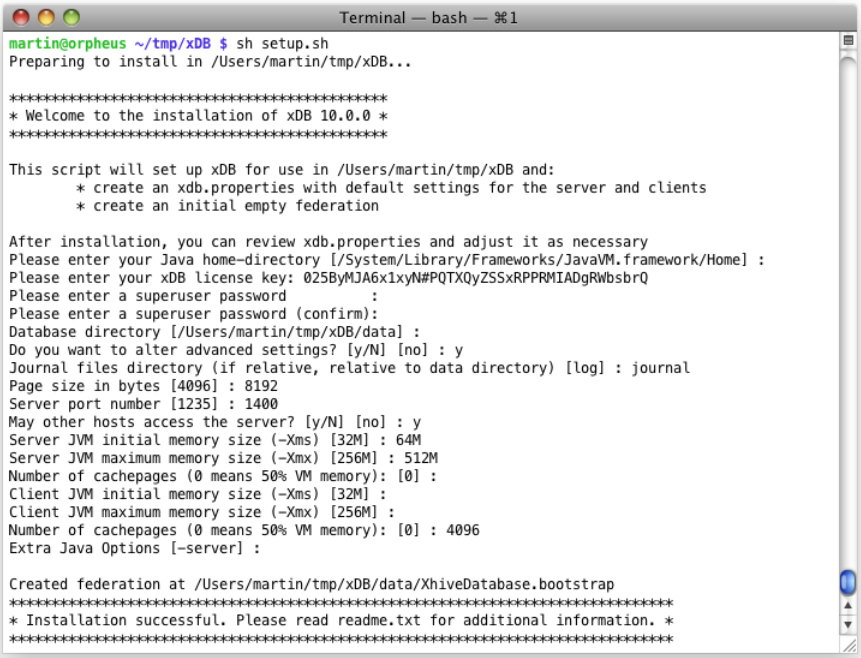
   Make sure that you have a working Java executable in your PATH. You can verify the Java version by running **java -version** from the command line. See also the installation requirements, page 33.

2. Run the **sh setup.sh** setup script.

   You need Write permissions in both the installation directories, the directory to which you untarred the distribution and the directory in which you create your initial federation.

   After the installation files have been copied, enter the parameters that are required during installation.

**Figure 2 xDB installation options for UNIX**




The xDB installation program provides a default value for most questions. To accept a default value, press the **Enter** key, otherwise type a new value and press **Enter** to override the default value. If the information you enter is incorrect, an error message displays and the question is asked again. For questions that require a Yes or No answer, the default choice is displayed in upper case. For example, if the choice is **[y/N]**, **N** is the default value.

During the installation process, provide the following values:

- A base directory path to the JDK.

  The installation program attempts to identify where the location of the JDK based on the **JAVA_HOME** and **PATH** environment variables. Verify that the directory really contains the full JDK.

- A valid xDB license key.
- A superuser password for the xDB database administrator.

  The superuser password must be typed twice and is displayed in clear text when running Java versions older than 6.

- The database directory for the data stored in the database.

3. Select whether you want to modify advanced settings, as follows:
   - If you select **no**, the installation is completed using the 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 or smaller than the block-size of the file system where the database is located. See page sizes, page 212 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 database clients cache locally in client memory.
- Other Java command-line options. Usually these options are not changed.

After the installation is complete, the installer will have saved settings to the configuration file conf/xdb.properties and created an empty federation at the path you have chosen.

After you have installed xDB on Unix, perform the following post-installation steps:

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

# Verifying the xDB installation

The most important commands are listed in xDB commands, page 42. The same functionality is available through the xDB API. Running the commands without any parameters, displays information about each command.

**Table 9 Important xDB commands**

| Command | Description |
|---|---|
| **xdb admin** | Starts the graphical administration client for maintaining databases, users and content. |
| **xhive-ant**, page 24 | Compiles and runs the samples and other programs. |
| **xdb create-database**, page 52 | Creates a new database. |
| **xdb delete-database**, page 135 | Removes an existing database. |
| **xdb create-federation**, page 154 | Creates a new empty federation. |
| **xdb configure-federation**, page 135 | Sets the superuser password and license key on a federation. |
| **xdb backup**, page 150 | Saves a federation to a backup file. |
| **xdb restore**, page 151 | Restores a federation from a backup file. |
| **xdb info**, page 135 | Displays debug information on currently open transactions and their locks. |

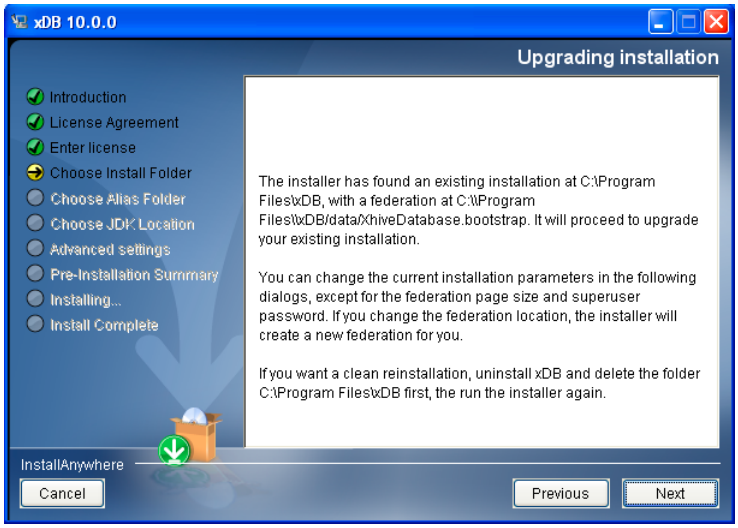
| Command | Description |
|---|---|
| **xdb run-server**, page 135 | Starts the dedicated server process for the default federation. |
| **xdb stop-server**, page 135 | Stops the dedicated server process for the default federation. |
| **xdb create-replica**, page 135 | Creates a full replica and optionally registers a replica id of a federation at a given location. |
| **xdb suspend-diskwrites**, page 135 | Ensures the federation files are flushed to disk, and suspends or resumes writing. |

# Upgrading an xDB installation

The Windows installer can automatically upgrade an xDB installation from version 9.0 or later to version 10 and later. It is not possible to upgrade an earlier xDB version using the installer - in that case, please manually uninstall the older database version and procede with a regular installation.

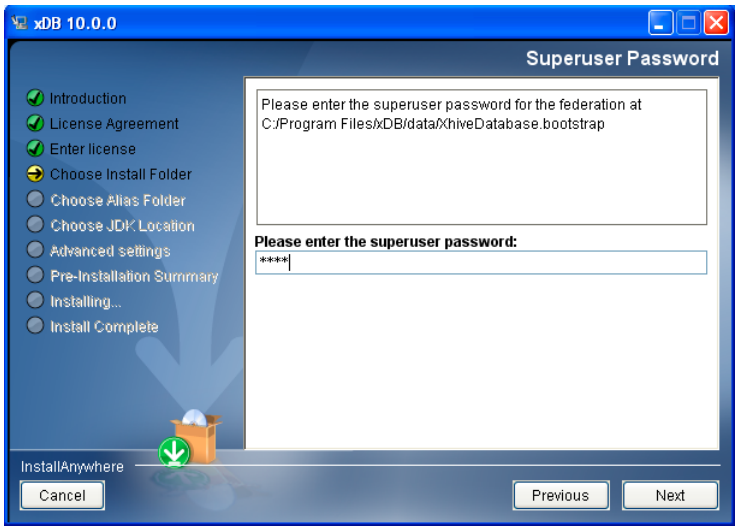
1. Execute **xdb_setup.exe** from the distribution and start with a normal installation.
2. When asked for the installation location, choose the same folder as you used to install the previous xDB version.

   The installer displays an upgrade notice.




3. Click **Next**.

   The installer asks for the superuser password for the existing federation.




4. Enter your superuser password. **WARNING:** Installation will fail if the superuser password is incorrect. Make sure that the superuser password is correct, as it will not be checked in this dialog and installation will fail if it is incorrect. Afterwards, click **Next**.

The installation process will continue as described in Installing xDB on Windows, page 34. Compatible settings from the previous installation will be adopted. You can change any of the advanced settings during the installation process, except for the page size and superuser password. If you choose a different federation location, the installer create a new federation instead of upgrading the old one. **Note:** The installer will remove the xDB 9.0 binaries, background process, documentation, and so on. If you want to install xDB 9.0 and 10.0 side by side, you have to choose a different installation location for each. It is not possible to have an xDB 9.0 and an xDB 10.0 background service configured by the installer. **Note:** The installation layout has changed slightly. Third party jar files are now in subdirectories within the lib directory, xdb.properties is moved to a subdirectory conf, and the Windows server will write logs to the subdirectory log.

# Configuration files

The xDB installation process creates three configuration files. These files are used by the tools shipped with xDB: the Admin client, page 127, the command line client, page 135, and the dedicated server process on Windows, page 48. The configuration files are **not** used by Java applications using xDB embedded through the Java API.

- $XHIVE_HOME\conf\xdb.properties

  This file contains settings that are used with the **xdb** command, the **xhive-ant** command, and the dedicated xDB server on Windows.

- $XHIVE_HOME\bin\xDB Server.lax

  This file contains JVM options for the NT service process on Windows. The file has the following options:

  – lax.nl.java.option.java.heap.size.initial. This option is equivalent to -Xms.

  – lax.nl.java.option.java.heap.size.max. This option is equivalent to -Xmx.

  – lax.nl.java.option.additional. This option changes JVM parameters.

– lax.nl.current.vm. This option changes the default JVM.

All other settings are read from the regular property file. This configuration only applies to Windows, on UNIX operating systems the server is configured using the values in the xdb.properties file.

- $XHIVE_HOME\bin\xDB Admin Client.lax

  This file contains JVM options for the **xDB Admin Client.exe** executable on Windows. These settings only apply when the administration client is started through the Start Menu shortcut or the executable instead of starting using the **xdb admin** or **xhive-ant run-admin** command.

The xdb.properties file contains the key/value pairs for the database settings described in the xDB settings table. Settings obtained from the configuration file can be overridden by setting an environment variable with the same name. Alternatively, the corresponding command-line switch can be passed to the tool.

The command-line tool first searches for a .xdb.properties file (note the leading '.'/dot) in the home directory of the current user. The .xdb.properties file can contain default xDB configurations for specific users.

**Table 10 xDB settings**

| Property | Explanation |
|---|---|
| XHIVE_BOOTSTRAP | The xDB 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 | The default user name for the command line tools. |
| XHIVE_PASSWORD | The default password for the command line tools. |
| XHIVE_MAX_MEMORY | Maximum memory used by a single command line tool, as in the `-Xmx` parameter to the JVM. |
| XHIVE_MIN_MEMORY | Minimum memory used by a single command line tool, as in the `-Xms` parameter to the JVM. |
| XHIVE_CACHEPAGES | The number of cache pages allocated in command line tools. If set to 0, half of the JVM memory is used. |
| XHIVE_FEDERATION | The location of the default database file. It is possible to have more than one federation. This property is used by the **xdb run-server**command (and the Windows service), the **xdb create-federation**, and the **xdb restore** commands. |
| XHIVE_SERVER_MAX_MEMORY | Maximum memory for the xDB server process. |
| XHIVE_SERVER_MIN_MEMORY | Minimum memory for the xDB server process. |
| XHIVE_SERVER_CACHEPAGES | Cache pages for the xDB server process. If set to 0, half of the JVM memory is used. |
| XHIVE_OPTS | Additional options to be passed to the JVM. |

| Property | Explanation |
|---|---|
| XHIVE_SERVER_ADDRESS | The server listens at this address. If set to "*****", the server accepts connections from all hosts, if set to "**localhost**" only connections from the same machine are accepted. |
| XHIVE_SERVER_PORT | The port used by the xDB server. |
| XHIVE_HOME | The installation location. It can be changed to use a different xDB 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 xDB tools. 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. |

Changing server-related settings, such as the XHIVE_SERVER_MAX_MEMORY property requires restarting the server. Other settings are applied the next time a command-line tool is run. For more information about the command-line tools, see Command line client, page 135.

# Creating a database

After xDB is installed, users can create databases and run the samples that are provided with xDB.

To create a database:

1. Run the **xdb admin** command from the command line or the Start menu in Windows to start the administration client. The command is located in the \bin subdirectory of the xDB target directory.
2. Select **Database > Create database** from the menu.
3. Create a database named **united_nations** with the superuser password as entered during the xDB installation and enter **northsea** as the administrator password. This database is used in all samples described in this manual.

**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.

## *Running a sample*

xDB samples can be run using the Ant tool.

All sample source files are located in the *XhiveDir*\src\samples\manual directory.

To run the samples:

1. Open a command prompt and navigate to the ***XhiveDir*\bin** directory.
2. Enter the following command:

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

The command 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.

# Configuring the xDB bootstrap

The xDB bootstrap is a URL pointing to a xDB federation, similar to the connection string in relational databases. For remote connections, it takes the form `xhive://hostname:port` or `xhives://hostname:port` for SSL encrypted connections. However in xDB, the bootstrap can also directly point to a file, usually called XhiveDatabase.bootstrap. In that case the database will run embedded in the current JVM. Only one process can run an embedded database for a particular federation.

Applications can explicitly specify the bootstrap they want to connect to by calling **XhiveDriverFactory.getDriver(String bootstrap)**. If an application calls the method without a parameter, **XhiveDriverFactory.getDriver()**, xDB will resort to envionment variables and properties to find a federation.

xDB will look in these places, in order:

- The Java system property xhive.boostrap
- The first line of a text file called xhive.bootstrap in the current working directory of the Java process
- The environment variable XHIVE_BOOSTRAP

The xDB command line tools will also use the environment variable XHIVE_BOOTSTRAP if no explicit federation argument is specified.

See also Connecting to a database, page 54.

## Examples for xhive.bootstrap

To connect to an xDB server remotely on a TCP/IP port, specify the xhive.bootstrap property as follows:

**java -Dxhive.bootstrap=xhive://*hostname*:*port***

To run a server in the current JVM, specify the xhive.bootstrap property, as follows:

**java -Dxhive.bootstrap=*PathName*/XhiveDatabase.bootstrap**

Where *PathName* is the complete path to the XhiveDatabase.bootstrap file. Enclose the parameter in quotation marks if it contains spaces. Depending on the application, running the server in the current JVM can be much faster than using a remote server. However, only one JVM can run an xDB server for a specific federation at the same time.

## API documentation

com.xhive.XhiveDriverFactory

# xDB Java Classpath

Using the Java command line instead of the xhive-ant script or developing xDB applications in an IDE, such as Eclipse, requires configuring the CLASSPATH variable to include the required JAR files, as described in Required JAR files below. The JAR file names do not include version numbers so that classpath entries do not need to be changed on minor version upgrades. The distribution includes a file lib/versions.txt that details library versions and licenses.

**Table 11 Required JAR files**

| Functionality | JAR files | Comments |
|---|---|---|
| Basic xDB | xhive.jar, google-collect.jar, xercesImpl.jar, xbean.jar, antlr-runtime.jar, icu4j.jar, aspectjrt.jar, lucene.jar, and lucene-regex.jar | The jsr173_api.jar file is not needed for Java 6 because those interfaces are already included in the Java 6 distribution. |
| XSLT transformations | xalan.jar, serializer.jar | If desired, another JAXP compliant XSLT processor can be used, such as Saxon. |
| PDF transformations using the XhiveFormatterIf interface | fop.jar, xmlgraphics-commons.jar, avalon-framework.jar, batik-all.jar, commons-io.jar, commons-logging.jar | |
| Command line clients | jline.jar | This JAR file is only used by the command line clients and is not necessary for applications using xDB. |
| xhive-ant script | ant.jar, ant-launcher.jar | These JAR files are only used by the xhive-ant script. They are not necessary to run applications. |
| Spring web applications | spring-context.jar, spring-webmvc.jar, spring.jar | These JAR files are used for the J2EE examples. |

# Using the xDB dedicated page server

During the xDB installation process, the system is configured with a dedicated page server. The pager server runs as a background process for other applications to connect to the database.

This background process is not a required part of the xDB architecture. Java applications can access the database directly through the bootstrap file without making any network connection. However, when accessing database files directly, no other processes, including the dedicated server, can access the database at the same time. If the main application starts a listener thread to which other applications can connect, accessing the database is not a problem.

The server program is a small Java program that accesses the xDB API. There are different configuration options, but the entire 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
```

Only the lines related to `startListenerThread` method are specific to accepting remote connections. If these code lines are added to an application, it can accept connections from other applications.

Running xDB with a separate dedicated server is not necessarily the best configuration for performance. Offering the separate server configuration as a default, is a more convenient way to get started. However, for production applications where performance is essential, we recommend running the complete database in process with the application itself.

## Configuring the dedicated server

The following server configuration values are determined during the xDB installation:

- Port number for accepting connection.

  The default is port number is 1235.
- Page-cache size

  The default page-cache size 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 can be modified 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. 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 configuration file changes, see the **xdb run-server** command and Configuring the server for SSL, page 156.

It is not required to use a server process to work with xDB. To configure xDB for use without a server:

- Use the /path/to/XhiveDatabase.bootstrap path instead of xhive://hostname:portname as the bootstrap path in your application.
- Stop the server. Only one process is allowed access to the federation. On UNIX, the server is not started if you do not run it. On Windows, stop the server using the **net stop xdb-server** command, and disable the automatic startup in the system preferences.

## Starting a background server process

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

On Windows, a background server is installed as a service. The service starts automatically when the Windows machine boots. Because the background server is implemented as a service, the SYSTEM account must have access to all involved resources, including JAR files and database files. You can start the server from the command line using **net start xdb-server**, and stop it using **net stop xdb-server**.

Windows also provides a Graphical User Interface to services. This server is set up during installation, so Windows users working with the default federation typically do not have to run **xdb run-server**.

On UNIX, due to the very different system layouts, xDB does not automatically configure a background service. Users are recommended to set up their own startup item, typically a shell script in `/etc/init.d`. The xDB server can be started as a background process using the typical shell syntax.

### Running an xDB server in the background on UNIX

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

## *Stopping a background server process*

The **xdb stop-server**command stops a running federation.

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

On Windows, the **net stop xdb-server** should be used, as Windows might try restarting a service that terminated otherwise, depending on the Windows configuration for the service.

# Uninstalling xDB

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

- Windows platform

  Use the Add/Remove Programs option in the Windows Control Panel or the `Uninstall xDB` link in the xDB program group to uninstall xDB. The uninstall process does not delete data directories, compiled samples, and other files created after the xDB installation. These directories and files can be removed manually afterwards.
- UNIX platform

  Use the **xdb stop-server** command to stop the page server if it is running, then remove the installation and data directories.

**Note:** The uninstalling.txt file contains detailed instructions for uninstalling xDB.

# Chapter 4

# Creating applications

This chapter contains the following topics:



## Building and running applications

xDB allows developing applications of varying complexities, using simple and advanced techniques, such as XPointer, XQuery, XLink and Abstract Schema.

There are samples for most of the important functions. The samples can be run using the xhive-ant tool. For example, running

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

compiles and runs the StoreDocuments sample. Before running the samples, adjust the values in the src/samples/manual.SampleProperties.java file to match the local system.

## Creating a database

By default, the xDB installation program automatically creates one federated database. This federated database acts as a place holder for other xDB databases. The federated database also contains the superuser information.

An xDB user can create as many databases as needed. A database can be created using one of the following:

- The xdb create-database command-line utility.
- The administration client.
- The Application Programming Interface (API) functions.

### Related topics

xDB architecture

## Creating a database using the command line

The **xdb create-database** command-line utility resides in the *XhiveDir*/bin directory.

To create a database:

1. Open a command line window and change to the *XhiveDir*/bin directory.
2. Run the **xdb create-database** command, as follows:

   ```
   xdb create-database -p SuperUserPassword --adminpwd
   AdminPassword DatabaseName
   ```

   Where **SuperUserPassword** is the password of the superuser account, **AdminPassword** is the password of the administrator account, and **DatabaseName** is the name of the database you want to create.

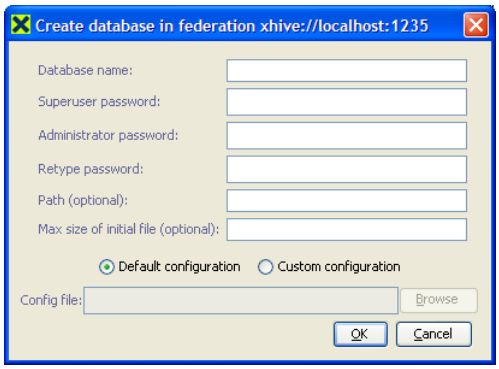
   **Note:** xDB database names, user names, and passwords are case sensitive. For example, Xhive and XHIVE are treated as different database names. Passwords must be alphanumeric and from 3 through 8 characters long.The default administrator password used in all samples is **northsea**. Make the appropriate changes to the SampleProperties.java file if you are using a different password.

## Creating a database using the administration client

To create a database using the administration client:

1. Start the administration by executing the **xdb admin** command in the \bin directory of the xDB installation. On Microsoft Windows platforms you can select the administration client in the Startup menu.
2. Select **Database > Create database**.

   The Create database window displays.




3. Enter the database name, superuser password, and administrator password, then click **OK**.

## Creating a database using the API

Log in as the superuser to use the API.

The **createDatabase()** method is part of the **com.xhive.core.interfaces.XhiveFederationIf** interface.

To create a database using the API:

1. Start a session and open a connection to the database 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);
```

   **Note:** By default, the CreateDatabase.java sample uses **northsea** as the superuser password. If you entered a different superuser password during xDB installation, change the source code of the sample accordingly.

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

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

3. Call the **createDatabase()** method with the name of the new database and the administrator password. You can specify a configuration file or use a default configuration, as follows:

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

   For more information about the physical file structure of a database, see Database files, page 29.
   **Note:** A superuser can only create and delete, but not administer a database. Log on as the administrator to manage the database.

### Samples

CreateDatabase.java

### API documentation

com.xhive.core.interfaces.XhiveFederationIf

# Connecting to a database

To connect to an xDB database:

1. Obtain an xDB driver, as follows:

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

   If you do not specify the bootstrap location, page 47 in the JVM environment, pass the location as an argument to the **getDriver()** method, as follows:

   ```
   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, as follows:

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

   Only initialize a specific driver once in your application. Use the **isInitialized()** method to verify whether the driver has been initialized.
3. Create a new **XhiveSessionIf** session, as follows:

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

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

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

### Samples

ConnectDatabase.java

### API documentation

com.xhive.XhiveDriverFactory

com.xhive.core.interfaces.XhiveDriverIf

com.xhive.core.interfaces.XhiveSessionIf

# Using JAAS

xDB provides user authentication, but also allows utilizing external authentication systems based on JAAS, such 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. 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 **DialogCallbackHander** class that the administration client uses for JAAS authentication.

The details of implementing and configuring JAAS are beyond the realm of this manual since they depend on the individual PAM server and authentication system specifics. For more information about JAAS, see the LDAP sample in the *XhiveDir*/src/samples/ldap directory. It describes the interfaces that must be implemented and the API calls that enable JAAS authentication.

Sun Microsystems offers a JAAS Reference Guide, JAAS Authentication Tutorial, and API documentation.

### Samples

../ldap/SampleClient.java

../ldap/XhiveServerWithLDAP.java

### API documentation

com.xhive.core.interfaces.XhiveDriverIf

com.xhive.core.interfaces.XhiveSessionIf

# Creating sessions

In xDB, all operations to databases take place within a session. The developer can determine the scope of a session.

To create a session in xDB:

1. Establish a connection to the database, as described in Connecting to a database, page 54.
2. Create a session using the **createSession()** method from the **com.xhive.core.interfaces.XhiveSessionIf** interface.

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

# Using transactions

One or more transactions can occur within a session. A transaction is 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. Either all the instructions are completed successfully or the transaction fails. The database is never left in an inconsistent state.

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 using either using the **commit()** or the **rollback()** method.

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

   After a call to **commit()** or **rollback()**, all references to database objects (such as nodes, libraries, etc.) become invalid. Use **checkpoint()** (see below) if you want to continue using the objects.

## Example

The following example describes a transaction that includes an application that parses external XML documents and appends them to a library. If an error occurs during parsing or appending, the complete transaction is rolled back and none of the files 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();
}
```

Apart from the **commit()** method, xDB also provides the **checkpoint()** method, which can be used to commit all persistent operations executed since the last **begin()** or **checkpoint()** method call. When using the **checkpoint()** method the transaction remains active after the **checkpoint()** call and references to database variables remain usable.

After performing a **commit()** call, you can perform a **begin()** call to start a new transaction on the same session. A **get()** call must be used on all database variables on the session. For example, the following code does not execute but throws an **XhiveException.OBJECT_DEAD** exception when the library is accessed after the second **begin()** call.

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
session.commit(); session.begin();
System.out.println(library.getName());
session.commit();
```

Another concurrent session has removed the referenced database objects. The solution is to get a new reference to the library after the second **begin()** call.

```
session.begin();
XhiveLibraryIf library = session.getDatabase().getRoot();
session.commit();
session.begin();
library = session.getDatabase().getRoot();
System.out.println(library.getName());
session.commit();
```

**Note:** Although a **disconnect()** call marks the end 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, it terminates the TCP connection to the xDB server. If the **terminate()** method is not called, the TCP connection is closed when the session is finalized, after it has been garbage collected.

### Samples

UseSessions.java

### API documentation

com.xhive.core.interfaces.XhiveSessionIf

com.xhive.core.interfaces.XhiveDriverIf

# Creating libraries

An xDB library stores documents or other libraries. Users can create multiple libraries to build a document hierarchy or storage architecture.

The nested structure of libraries within xDB is like the nested structure of directories or folders within a file system. Any library can contain other libraries and there is exactly one root library. The root library is created automatically with a new database.

To create a library:

1. Obtain a handle to the parent library. If the root library is the parent 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. This step is optional.
4. Append the new library to its parent using the **appendChild()** method.

## Example

The following sample code creates a library structure in the sample database with one top-level library called `Publications`. This top-level library has 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**


xDB also supports detachable libraries, as described in Managing detachable libraries, page 152. Creating a detachable library requires calling the **XhiveLibraryIf.createLibrary(int options, String segmentId)** method with the **XhiveLibaryIf.DETACHABLE_LIBRARY** option flag. When the XhiveLibraryIf.DETACHABLE_LIBRARY flag is set, a segment ID must also be specified. Moreover, the segment must have been created and not have been used. Once a segment is used for a detachable library, it can be used for that library only, and cannot be shared with other detachable libraries.
**Note:** Although naming a library is optional, it is recommended as several access and indexing methods only work with named libraries.

## 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

# Library metadata

All library children, such as libraries, documents and blobs can contain metadata. Metadata consists of key/value string pairs that are used to store information about a document. The metadata is stored in a java.util.Map file that can only contain strings, for example:

```
XhiveDocumentIf doc = ...;
doc.getMetadata().put("author", "Jane Doe");
```

If the document is versioned, the metadata cannot be changed. When a new version of the document is checked in, the metadata of the new version overrides the metadata of the previous document version.

For more information about using metadata in XQuery queries, see the **xhive:metadata** function in XQuery options and extension expressions, page 167.

# DOM configuration settings

The **DOMConfiguration** DOM level 3 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** function of the **XhiveDocumentIf** interface uses these settings. The **XhiveDocumentIf**, **LSParser** and **LSSerializer** objects each have their own configuration object.

The complete list of options can be found in the JavaDOC documentation of **DOMConfiguration** interface, and the **getDomConfig()** method of the **LSParser** and **LSSerializer** objects.

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

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

## Specification deviations

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 have a negative impact on query and validation performance. Therefore the **LSParser** default value is set to **"Boolean.FALSE"** in xDB.

## Additional parameters

xDB includes several parameters for the **LSParser** and **Document** objects.

**Table 12 Additional parameters**

| Parameter name | Interface | Default value | Description |
|---|---|---|---|
| xhive-ignore-catalog | XhiveDocumentIf, LSParser | Boolean.FALSE | Specifies whether the corresponding DTD's and XML schemas are ignored in the catalog. |
| 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". | Instructs the server 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. |
| 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 | 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.<br><br>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. |

| Parameter name | Interface | Default value | Description |
|---|---|---|---|
| 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 | 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.<br><br>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 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

# Storing BLOBs

xDB can store Binary Large Objects (BLOBs) within a database. BLOBs are typically image files, sound files, and Microsoft Office files. Storing BLOBs and XML documents in the same database allows managing all resources for a specific project or product in one uniform way.

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

# Importing non-XML data

xDB can import data from non-XML files. The **com.xhive.util.interfaces.XhiveSqlLoaderIf** interface contains the methods used for importing non-XML data. For a detailed specification of these methods, see the **XhiveSqlLoaderIf** Javadoc.

### Example

The following example describes how to import data in CSV format into xDB and store the data as an XML document.

The example uses the following 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 data is imported using a **XhiveSqlLoaderIf** object and the `loadSqlData()` method:

```
Document un_members_doc = loader.loadSqlData(Impl, FileName, ',', '\\', '"',
            true, "UN_members", XhiveSqlLoaderIf.IGNORE_HEADER, "member",
            new String[] {"name","admission_date","additional_note"},
            new boolean[] {false, false, false});
```

The example code specifies the following information:

- DOM implementation:`Impl`.
- File name: variable `FileName`.
- Separator: `,`
- Escape character: `\`
- Incloser: `"`. The incloser is the character that specifies the start and end of a string.
- Column names: `true`. The first row in the data file contains column names and not data.
- Document element name: `UN_members` specifies the name of the root element.
- Mode: `IGNORE_HEADER` specifies that column headers are not used as element names.
- Row element name: `member` is the name of the element used to enclose each row.
- Column element names: `new String[] {"name", "admission_date", "additional_note"}` specifies the names to use for each of the elements.

- Column element ignore: `new boolean[] {false, false, false});` specifies whether to ignore columns.

The example code generates 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>
```

The **loadSqlData()** method can be used to import data from a JDBC-compliant relational database. For more information about the **loadSqlData()** method, see the API documentation.

### Samples

StoreRelationalData.java

### API documentation

com.xhive.util.interfaces.XhiveSqlLoaderIf

# 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.XhiveNodeIf

org.apache.xerces.xs

# 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.

These basic security and authorization levels use the xDB authority object in the xDB **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:

- 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.

The authority object offers different user access types and the access level for each type can be customized:

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

The xDB API includes several methods for managing users, user lists, groups, and group lists, as described in Management interfaces and methods, page 66.

**Table 13 Management interfaces and methods**

| Interface | Methods |
|---|---|
| XhiveUserListIf | addUser(), removeUser(), getUser(), hasUser() |
| XhiveUserIf | setPassword(), isAdministrator() |
| | 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);
}
```

For more information about the interfaces, see the API documentation.

### Samples

ManageUsers.java

### API documentation

com.xhive.core.interfaces.XhiveGroupIf

com.xhive.core.interfaces.XhiveGroupListIf

com.xhive.core.interfaces.XhiveUserIf

com.xhive.core.interfaces.XhiveUserListIf

# Using versioning

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

In the following example, the `makeVersionable()` method is used 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 a 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

# Chapter 5

# Configuring Multiple Backend Servers

This chapter contains the following topics:



## xDB multi-node architecture

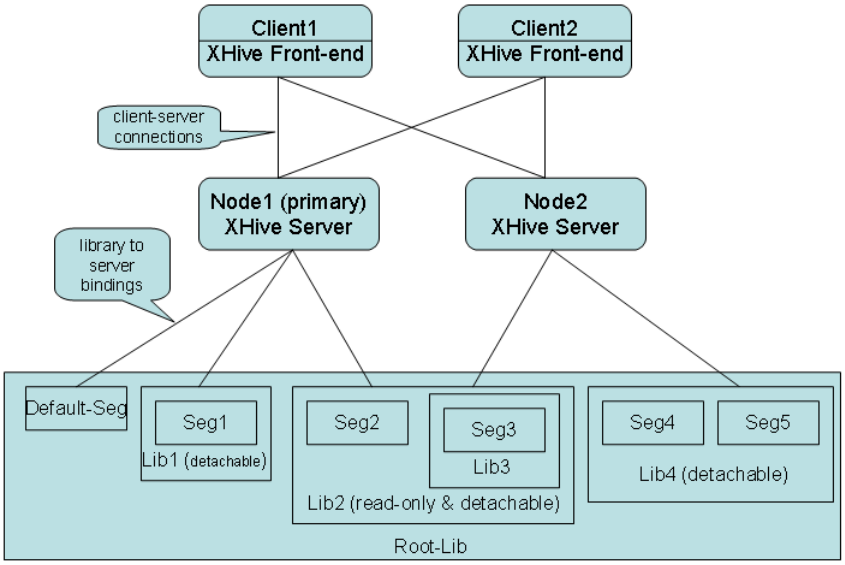
When running in client-server mode, xDB allows concurrent clients but only one single backend server per database. As the number of clients increases a single backend server can become a bottleneck. In an xDB multi-node architecture, multiple backend servers can run on different hosts to serve a single database. This configuration improves the horizontal scalability of the backend server without replicating data.

In a multi-node configuration, a backend server instance is called a *node server*. In this document, a node is defined as an xDB backend server instance. One of the node servers is designated as primary server and the other node servers are called non-primary servers. Each node server can run on a separate host, or multiple node servers can run on a single host. An xDB multi-node configuration requires that xDB is used in client-server mode. The primary server must always be active to access the database.

xDB does not support a two-phase transaction commit protocol for managing distributed transactions. Instead, one node server manages a separate set of data in a database through data partitioning and binding. Detachable libraries, page 30 provide a natural data partitioning mechanism. A data binding mechanism is used to assign the detachable library to a node server. Any reading or writing of the detachable library data pages must go through the binding node server.

The xDB multi-node configuration, page 70 describes a two-node configuration. Node server 1 serves the root library, lib1, lib2, and lib3. Node server 2 serves lib2, lib3, and lib4.

**Figure 4 xDB multi-node configuration**




A multi-node architecture based on data partitioning and binding imposes several constraints:

- Data storage must be symmetrically accessible to all the server hosts. This restriction 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.
- An update transaction can make updates only to libraries bound to a single node. However, the update transaction can read data pages in any libraries of the database as long as it does not violate the locking rules, page 77. Transactions must follow the locking rules to avoid distributed deadlock.

## Example

The following example describes transactions T1, T2, and T3, that occur between the libraries in the xDB multi-node configuration, page 70.

| Transaction 1 (T1) | Transaction 2 (T2) | Transaction 3 (T3) |
|---|---|---|
| begin | begin | begin |
| . . . | . . . | . . . |
| update Root-Lib | read Root-Lib | update Lib1 |
| update Lib1 | read Lib2 | update Lib4 |
| . . . | read Lib3 | . . . |
| commit | . . . | commit |
| | commit | |

T1 is valid because both, the root library and Lib1, are bound to the same server. T2 is also valid because no updates are done in the transaction. T3 is not allowed because it updates Lib1 and Lib4 that are bound to different nodes.

# Multi-node configuration

A multi-node configuration requires a modified bootstrap file format that includes information about the nodes, node server log directory, and the segment-to-server binding.

## Example

The following sample bootstrap file defines the configuration described in xDB multi-node configuration, page 70. There are two nodes, Node1 and Node2. The federation consists of six segments. The node configuration and segment to server bindings are highlighted. The name="primary" attribute for Node1 indicates that Node1 is the primary server. The default segment and Seg1 are bound to Node1. Segments Seg4 and Seg5 are bound to Node2. The other two read-only segments Seg2 and Seg3 are bound to both Node1 and Node2.

```
<?xml version="1.0" encoding="UTF-8"?>
<server version="xDB 9.1" 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">
     <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">
     <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">
     <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">
     <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">
     <file path="Shanghai-Seg4-0.XhiveDatabase.DB" id="4"/>
     <binding_server name="Node2"/>
  </segment>
  <segment id="Seg5" temp="false" version="1" state="detached"
   usage="detachable">
     <file path="Shanghai-Seg5-0.XhiveDatabase.DB" id="5/>
```

```
    <binding_server name="Node2"/>
  </segment>
</database>
</server>
```

# <node/>

The <node/> element describes a node server.

## Attributes

| Name | Default | Description |
|---|---|---|
| node name | | The name of the node server.<br>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.<br>The default log directory of primary server is **log**. The default log directory for a non-primary node server is *host name*-log. With multi-node support, the "log" element is a child of the "node" (instead of the "server") element.<br>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 , page 31 element in the bootstrap file.

## Attributes

| Name | Default | Description |
|---|---|---|
| name | | The name of the node server to which the segment is bound. |

# Library binding

Data to node server binding is implemented at the library level. When a library is bound to a node server, only the node server can read and modify the library. Though 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.

xDB enforces the following binding rules:

- A library is always bound to at least one server.
- The root library of a database is always bound to the primary server and the binding cannot be changed.
- A read-write detachable library can be bound to only one server.
- A read-only detachable library can be bound to more than one server.
- Binding a detachable library to a 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 by modifying the binding of the detachable ancestor.
- Only the database administrator can change the bindings of a detachable library.

## *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() method of XhiveDatabaseIf interface.

### Example

The following example creates a segment using the createSegment() method.

```
session.connect(dbaUserName, dbaUserPassword, "MyDatabase");
session.getDatabase().createSegment("node1", "segment1", null, 0);
```

Once a detachable library has been created, one can use the changeBinding(), addBinding(), and removeBinding() methods of the XhiveLibraryIf interface to change, add, and remove the bindings of the library.

The following example changes the binding of a library to the node **Node1**.

```
XhiveLibraryIf library = session.getDatabase.getByPath("/library1");
library.changeBinding("Node1");
```

# Managing nodes

xDB automatically creates a primary node as part of a federation. The primary node cannot be removed. Non-primary nodes can be added or removed by calling the addNode() or removeNode() methods of the XhiveFederationIf interface. The updateNode() method updates an existing node. Non-primary servers can be shut down using the shutdown() method

The following 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.

### 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 example uses the **xdb add-node**, **xdb remove-node**, and **xdb update-node** commands to add, remove, and update node servers. Note that these xdb commands call the corresponding API methods of XhiveFederationIf respectively.

```
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
```

The following example 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());
  }
```

# 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, xDB 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.

# High-availability support

There are several scenarios that require replacing a host machine. For example, a hardware failure, server crash, or to install a faster server to improve performance. The replacement procedure varies depending on whether the primary or a non-primary server is replaced.

The procedure can involve replacing the entire node, or keeping the node and replacing only the underlying host machine. The latter process is also called a *node identity change*.

For a description of the replacement procedures, see


- Changing node identity, page 76.
- Replacing a non-primary server, page 75.
- Replacing a primary server, page 76.


## *Replacing a non-primary node*

This task describes how to replace a non-primary server in a multi-node configuration. The procedure replaces node N1, running on host H1, 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 but 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 procedure 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 describes how to launch a backend server as the primary server. Specifying the node name **primary** as the second argument in the getDriver() methods establishes the server instance as the primary server.

```
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.

# 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 the 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.

xDB provides methods that enforce the locking rules and detect distributed deadlocks. The enabling distributed deadlock detection, page 77 task describes how to enable checking for distributed deadlocks.

## *Enabling distributed deadlock detection*

xDB can enforce locking rules, page 77 to avoid distributed deadlock. Distributed deadlock detection is enabled by default. With the deadlock check mechanism enabled, xDB throws an exception when a transaction violates the locking rules.

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

# 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. When 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.

# Replication

Replicating a multi-node configuration is currently not supported.

# Server upgrade

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 existing federation is changed into a multi-node configuration, the administrator has to set up each node manually.

# Chapter 6

# Catalogs and Validation

This chapter contains the following topics:



# 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 also called local catalogs. They can be created by calling the **addLocalCatalog** method. Local catalogs override information in the root catalog and during queries the local catalog is searched first. If a root and a local catalog contain a model with the same Id, 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 Id that depends on the schema type of the model:

- A public ID identifies a DTD model. If the DTD does not contain a public Id, xDB automatically generates an Id.
- A file name identifies an XML schema.

## API documentation

com.xhive.dom.interfaces.XhiveLibraryIf

com.xhive.dom.interfaces.XhiveCatalogIf

org.w3c.dom.as.ASModel

# 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 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 a model. For more information, see the Abstract schema , page 103code example.
- By parsing and validating a document. During validated parsing, the model linked to the document is automatically stored in the catalog.

# 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 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, as described in Normalizing XML documents.

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.

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.

# 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 103 interfaces contain the methods for validating documents that use a DTD.

### XML schema

The configuration, page 59 settings of the document determine the XML schema validation process, as described in Normalizing XML documents. 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

# Chapter 7

# Managing Documents

This chapter contains the following topics:



## Creating and managing documents

xDB databases store XML data as documents using the **org.w3c.dom.Document** xDB API interface. This interface includes methods for creating an XML document, updating XML documents, and accessing document elements, comments, and attributes.

## 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, as follows:

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

### Samples

CreateDocument.java

### API documentation

org.w3c.dom.Document

org.w3c.dom.DOMImplementation

# Storing XML documents

The **appendChild()** is used to store XML documents in an xDB database. The **appendChild()** method requires a handle to the library where the document is going to be stored, for example the root library. The following example describes how to store a document in the root library of the sample database:

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

Documents can also be stored using the standard **insertBefore()** DOM method, like the following:

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

Where `firstDocument` specifies the name of the new document and `secondDocument` specifies the name of the document in front of which the new document is stored.

### Samples

StoreDocuments.java

### API documentation

org.w3c.dom.Node

# 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, the application checks whether the catalog contains an abstract schema that matches either the DTD or the XML schema. The application uses the abstract schema instead of the external DTD. Catalog checking can be turned of 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

# Parsing XML documents

Generally, documents are parsed when they are validated against a DTD or XML schema, or when they are imported from an external source. xDB utilizes the **parseURI** method of the **LSParser** DOM interface.

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

xDB supports the DOM Load and Save specification, which provides a standard for parsing and serializing DOMs.

For more documentation, see the W3C Document Object Model (DOM) Level 3 Load and Save Specification.

### Example

The following code example describes how to use the `parseURI` method 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());
```

Storing a parsed document in the database requires calling the **appendChild** method. Otherwise, the document is only parsed and 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

## *Parsing documents with context*

Documents can be parsed with context using the **parseWithContext** function on the **LSParser** method, as described in the following code example:

```
LSParser parser = charterLib.createLSParser();
// Using (null, null, null) as arguments means the document is 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);
```

The code example also creates the document and sets indexes, before parsing the data on it. There can be a performance gain for large documents, if the document is indexed during parsing.

### Samples

DOMLoadSave.java

### API documentation

org.w3c.dom.ls.LSParser

org.w3c.dom.ls.LSSerializer

com.xhive.dom.interfaces.XhiveNodeIf

# Retrieving documents and document parts

xDB offers several ways to retrieve documents and document parts from the database. For example, documents can be retrieved using DOM operations, by document Id and document name, using the **executeFullPathXPointerQuery()** method, XQuery, XPath, or using an index.

### Samples

RetrieveDocuments.java

RetrieveDocumentParts.java

### API documentation

org.w3c.dom.Node

org.w3c.dom.Element

com.xhive.dom.interfaces.XhiveLibraryChildIf

com.xhive.dom.interfaces.XhiveLibraryIf

com.xhive.dom.interfaces.XhiveNodeIf

## *Retrieving documents 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. The DOM operations for retrieving documents, page 89 table describes several of the DOM operations.

**Table 17 DOM operations for retrieving documents**

| Interface | Methods | Description |
|---|---|---|
| org.w3c.dom.Node | getFirstChild(), getLastChild(), getPreviousSibling(), getChildNodes() | Methods for retrieving document children. |
| | getNextSibling(), hasChildNodes() | Methods for retrieving specific document parts. |

| Interface | Methods | Description |
|---|---|---|
| org.w3c.dom.Element | getAttribute(), getAttributeNode() | Methods for retrieving element attributes within a document. |
| com.xhive.dom.interfaces.XhiveNodeIf | getFirstChildByType(), getFirstChildElementByName(), getDescendantByKey() | Methods for retrieving document children by type, name, and key. 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 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();
  }
}
```

# Retrieving documents by 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());
```

# Retrieving documents by document name

Although every document has an identifier, it can be more convenient to retrieve documents by their name. Document names are optional but have to 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 );
```

# Retrieving documents by 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 FullPathXPointerQuery relative to the new sub library
docsFound = newLib.executeFullPathXPointerQuery("../UN Charter - Chapter 3");
docRetrievedByFPXPQ = (Document)docsFound.next();
System.out.println(docRetrievedByFPXPQ.toString());
```

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

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

# Using an XPointer expression

If the library path contains an XPointer 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]][#xpointer_query]
```

Optional parameters are enclosed in [].

Adding an XPointer expression to the query string allows using the **executeFullPathXPointerQuery()** 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 queryXPointer = "#xpointer(/descendant::title)";

Iterator resultNodes = rootLibrary.executeFullPathXPointerQuery(
 sampleDocPath + queryXPointer);
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.

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

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

The **executeFullPathXPointerQuery()** 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 queryXPointer = "#xpointer(/descendant::title)";

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

For more information about XPointer queries, see Retrieving documents using XPath and XPointer, page 93.

# Retrieving documents 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 ...
}
```

For more information about XQuery, see Using XQuery, page 159.

## Samples

XQuery.java

## API documentation

com.xhive.dom.interfaces.XhiveLibraryChildIf

com.xhive.query.interfaces

# Retrieving documents using XPath and XPointer

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

# Using XPath

XPath queries can be executed using the **executeXPathQuery(...)** methods in the **XhiveNodeIf** interface. An XPath query can contain namespace declarations, variable and function bindings, and an absolute root. The results of an XPath query are returned from the **XhiveQueryResultIf** interface.

The **XhiveQueryResultIf** interface includes methods for extracting different types of information from a query result. A result can be a string, a Boolean, a number or a location set. A location set is a collection of nodes, points, and ranges.

The following methods can be used to extract different result types:

- **getStringResult()** - Retrieves the string value of a result.
- **getBooleanResult()** - Retrieves the Boolean value of a result.
- **getNumberResult()** - Retrieves the numeric value of a result.
- **getLocationSetValue()** - Retrieves the location set value of a result.

For more information about the conversion rules for these methods, see the XPath specifications.

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

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

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

XPathXPointerNamespaces.java

### API documentation

com.xhive.query.interfaces

com.xhive.xpath.interfaces.XhiveXPathContextIf

com.xhive.dom.interfaces.XhiveNodeIf

## Retrieving documents using indexes

Using indexes can dramatically improve query performance. For more information about indexes, see Indexes, page 111.

# 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 96
- Function objects, page 98

### 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

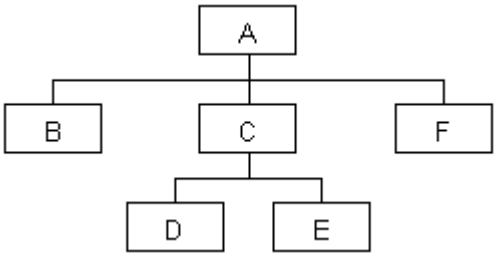
```
<A>
 <B>some text</B>
 <C>
  <D>1st child of C</D>
  <E>2nd child of C</E>
 </C>
 <F>some more text</F>
</A>
```

is associated with the flat representation

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

and the tree representation.

**Figure 5 XML document tree**




The interfaces and methods used for traversal are described in the Traversal interfaces and methods, page 97 table.

**Table 18 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:<br>• **FILTER_ACCEPT** - The current node is included.<br>• **FILTER_SKIP** - The current node is not accepted, but the children of the current node are considered for acceptance.<br>• **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

XML documents can be published from xDB, using the following interfaces:

- **XhiveTransformerIf**

  This interface enables publishing using XSLT.
- **XhiveFormatterIf**

  This interface enables publishing to PDF format.

**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.

### Samples

Publish2HTML.java

Publish2PDF.java

### API documentation

com.xhive.util.interfaces.XhiveTransformerIf

com.xhive.util.interfaces.XhiveFormatterIf

## Publishing XML documents using 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, WAP/WML, PDF, or any other format. 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 XSL document, which is actually an XML file, specifies the transformations that produce the desired output. The output can be another XML document, or a document of any other format.

### Example

The following example parses an XSL file, uses the `transformToString()` method to transform a document, and outputs it as a string.

```
// parse the XSL source file
LSParser parser = charterLib.createLSParser();
Document xslDocument = parser.parseURI(new File(
 baseDirectory + "/publish2HTML.xsl").toURL().toString());

// get a handle to the Transformer implementation
XhiveTransformerIf transformer = XhiveDriverFactory.getDriver().getTransformer();

// retrieve the document to publish
Document firstDocument = (Document) charterLib.get("UN Charter - Chapter 1");

// transform the XML document using the XSL document
String result = transformer.transformToString( firstDocument, xslDocument);
```

## Publishing documents to PDF

XML documents can be published to PDF format with the **formatAsPDF()** method in the **com.xhive.util.interfaces.XhiveFormatterIf** interface.

The formatAsPDF() method can format an XSL-FO document or an XML document as a PDF string. Transforming an XML document also requires an XSL document.

### Example

The following example applies the `formatAsPDF()` method to an XML document and a parsed XSL document to produce a PDF string that is written to a file.

```
// parse the XSL source file
LSParser parser = charterLib.createLSParser();
Document xslDocument = builder.parseURI(new File(
 baseDirectory + "/publish2PDF.xsl").toURL().toString());

XhiveFormatterIf formatter = XhiveDriverFactory.getDriver().getFormatter();

// format the XML document as PDF using the parsed XSL document
String result = formatter.formatAsPDF(rootLibrary, firstDocument, xslDocument);

// output the PDF-file
File pdfFile = new File(baseDirectory + "/output.pdf");
FileWriter fw = new FileWriter(pdfFile);
PrintWriter pw = new PrintWriter(fw);
pw.println(result);
fw.close();
```

# Linking documents

XLink is a W3C recommendation that enables links between XML documents. XLink creates simple and complex links. For more information about XLink, see the W3C XML Linking Language (XLink) Version 1.0 documentation.

All XLink information is stored in attributes. For example, 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.

The XLink information is accessible through the DOM API because it is stored as attributes. xDB provides methods and interfaces in the **com.xhive.dom.xlink.interfaces** package that are more convenient and easier to use. A selection of these methods is described in XLink methods, page 102 table.

**Table 19 XLink methods**

| Methods | Description |
|---|---|
| getTitle (), getHRef(), getRole() | Retrieve specific attributes. |
| getLinks(), getLinksByTitle(), getLinksByRole() | Retrieve all available links, by title, or role. |
| getNodesLinkingTo() | Retrieves all nodes linking to a specific resource. |
| getArcs(), getFrom(), getTo(), getStartingResources(), getEndingResources() | Retrieve all information from arcs. |

| Methods | Description |
|---|---|
| getLocators(), getRole(), getLabel() | Retrieve all information from locators. |
| expandDocument(), getResourcesLinkedBy() | Retrieve the content of the targeted resources. |

### Example

The **DomLinkBase.java** sample describes several of the XLink methods available in xDB.

```
//Gets an Iterator with all extended links
for (Iterator i = linkBase.getLinksBy("type", "extended"); i.hasNext();) {
  XhiveExtendedLinkIf link = (XhiveExtendedLinkIf) i.next();
  System.out.println(" Title = " + link.getTitle());
  //Gets an Iterator with all arcs
  for (Iterator j = link.getArcs(); j.hasNext();) {
    XhiveArcIf arc = (XhiveArcIf) j.next();
    if (arc.getFrom().equals("Margaret Martin")) {
for (Iterator m = arc.getEndingResources(); m.hasNext();) {
  XhiveLocatorIf locator = (XhiveLocatorIf) m.next();
  System.out.println("   *Checked out resource = " + locator.getLabel());
}
    }
  }
}
```

### Samples

XLink.java

### Samples

DomLinkBase.java

### API documentation

com.xhive.dom.xlink.interfaces

# 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 ASModels for DTDs or XML schemas in a catalog. xDB stores the abstract schema of document during validated parsing. The setActiveASModel() method uses an abstract schema that has already been stored in the catalog, as described in the following example.

```
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 describes 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();
```

### Samples

DOMValidation.java

### API documentation

org.w3c.dom.as

com.xhive.dom.interfaces.XhiveCatalogIf

# 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.

# Retrieving previous document versions

There are several methods to retrieve document versions, as described in the Retrieval interfaces and methods, page 105 table.

**Table 20 Retrieval interfaces and methods**

| Interface | Method | Description |
|---|---|---|
| XhiveLibraryChildIf | getXhiveVersion() | Returns the last version of a document. |
| XhiveVersionIf | getPreviousVersion() | Returns the previous version of a document. |
| XhiveVersionIf | getNextVersions() | Returns all subsequent versions of a document. |
| XhiveVersionIf | getVersionSpace() | 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

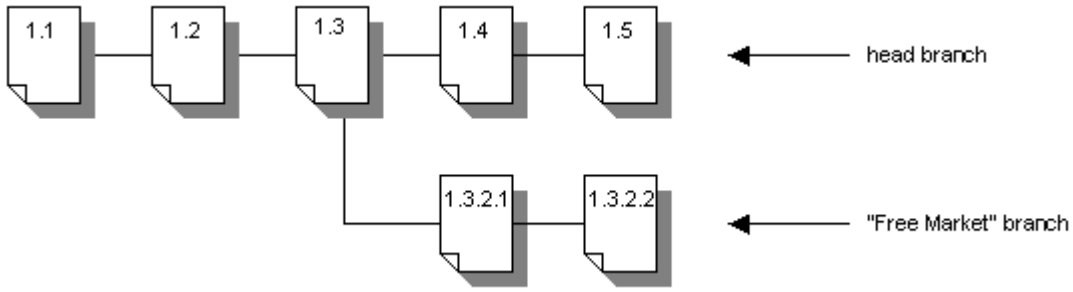
A version branch is a sequence of versions which has been separated from the main sequence of versions. Branches are typically used when multiple users are working in parallel on the same

documents. If a document is checked in and the version has one or more successors, xDB automatically creates another version branch.

**Note:** A branch can be created manually by calling the **createBranch()** method in **XhiveVersionIf** interface.

In the example described in Branching documents, page 106, 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 6 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, ...

xDB includes several methods for retrieving branch and version information, as described in the branch and version methods, page 106 table.

**Table 21 Branch and version methods**

| 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 external editors

xDB provides FTP-and DAV services that can be used with FTP or DAV aware programs. These services allow utilizing external document editors. Both of these services include the full source code, so they can be modified to match any environment.

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

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

The WebDAV (DAV) server is based on the Tomcat 4.x DAV-server. Installation instructions are included in the Tomcat distribution. After starting one of these servers, they can be accessed using FTP and DAV-clients, such as:

- **Windows Network Places**

  Network places for both FTP-sites and DAV-URLs can be added under the Windows Network Places folder. After starting one of the servers, the contents of the database displays as a folder. Documents can be added to the database using drag & drop. The documents can be viewed in an editor, but cannot be modified.
- **XML Spy**

  XML Spy reads and writes documents on both FTP- and DAV servers.
- **XMetaL 4.0**

  XMetaL 4.0 works with release 1.5 or later of the xDB WebDAV server. XMetaL 3.0 supports WebDAV, but is incompatible with Apache's WebDAV and the xDB DAV implementation.
- **Ultraedit**

  Ultraedit supports FTP-editing.
- **MS Word**

  MS Word reads and writes binary files from a DAV-server.

- **Webdrive**

  WebDrive maps FTP- and DAV-servers to drive letters. There are some issues using this tool in combination with the xDB/services. We recommend turning off 'Cache Files' in the program settings.

For Windows Network Places and XML Spy, the URLs for accessing an FTP-server has the format `ftp://localhost/`. The URL for accessing a DAV server with standard Tomcat settings has the format `http://localhost:8080/xhivedav`, where *localhost* is the host name of the machine running the server.

# 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 the following:

- Installing and starting the WebDAV server.
- 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.
- Saving the document to the WebDAV server.

## 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 through WebDAV, for example `http://localhost:8080/xhivedav/xhive-catalog/play.dtd`.

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

```
<!DOCTYPE rootElem PUBLIC "X-Hive/DB public id" "xhive-catalog/filename.dtd">
```

# Troubleshooting DTDs

In xDB, 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. xDB makes the catalog 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 a DAV/ 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 is set when the document is parsed for the first time using the **XhiveDocumentIf.fixupDoctypeIds** method.

The following information can be helpful when troubleshooting DTDs:

- xDB does not store the DTD when FTP or DAV 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.
- We recommend using FTP instead of WebDAV whenever possible. FTP is a simpler protocol and the full implementation is in one class file that does not depend on third-party solutions.
- The WebDAV protocol requires that the file size of each document is known at all times. The file size of XML documents can only be determined by serializing the file. Serializing is a slow process, especially when a directory listing is retrieved. xDB caches previously determined file sizes and shows a default file size of 2 KByte.
- When documents are parsed into the database, or a remote client creates new libraries, they are parsed or created with certain default parameters. The default parameter can be changed in the source code of the servers.
- When storing documents on the FTP/ WebDAV-server, new documents are created during parsing and existing documents are replaced. When using non-live indexes like Context Conditioned Indexes (and pre-X-Hive/DB 4.0 FTI indexes), code is added to the FTP/ WebDAV sources to update those indexes.
- When documents or DTDs are stored in the database over FTP/ DAV, 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 and DAV servers 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 xDB FTP server cannot run on the default port number 21. The port-constant must be changed to connect the xDB FTP server to another port.
- DAV folders should not be renamed. The current DAV-server implementation creates a folder, copies all the documents to the new folder, and deletes the other folder. The administration client or the FTP-server can be used to rename folders.
- When using the combination WebDrive, XMetaL and DTDs, XMetaL wants to store BLOBs (`dtdname.rlx`) in the catalog. xDB does not allow this operation. The internal WebDAV connection of XMetaL 4 works directly with xDB.

# Chapter 8

# Managing Indexes

This chapter contains the following topics:



## Indexes

Indexes are used to speed up queries. Especially with large data sets, indexes are essential to query performance.

An index stores key-value pairs. In xDB, the index key is a string, or in case of value indexes, a number type and a node set value. xDB currently supports the following indexing methods:

xDB indexes are updated automatically when the indexed data is updated. The only exception to that rule are context conditioned indexes which are deprecated and are discussed only in their specific manual section. Since updating the data means updating the indexes, the number of live indexes directly impacts the update performance.

The following restrictions apply to indexes:

- indexes can be defined for libraries or documents and are maintained automatically for all descendants and children of the library or document.
- Library name and library ID indexes can only be defined for libraries.
- An index can be accessed using an index list.

All index types use the **XhiveIndexIf** interface. Indexes are created with the following default properties:

- The index keys are always sorted.
- The index is stored in a separate storage container and is not locked with the library or document to improve concurrent access to the index.
- The key type of value indexes uses the **XhiveIndexIf.TYPE_STRING** key type.

Indexes are scalable and the number of index keys and values can grow without limitations. The node sets can become large, especially in case of ID attribute, element name, and value indexes.

Using indexes in XQuery, page 176 explains how to use indexes to speed up XQueries.

## Samples

FTI.java

CreateMultiPathIndex.java

LibraryIndexes.java

IdAttributeIndex.java

ValueIndexIndex.java

ElementNameIndex.java

IndexAdder.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

# 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.

## Examples

The following table describes various index options.

**Table 22 Index option examples**

| Index option | Description |
|---|---|
| `//elem` | Indexes all elements named **elem**. This index option is similar to an element value index. This type of 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 use 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. |

| Index option | Description |
|---|---|
| `//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"]`. |

### API documentation

com.xhive.index.interfaces.XhiveIndexIf

# Path index specification

Path indexes use following syntax:

| | | |
|---|---|---|
| spec | := | elementpath ( type \| '[' values ']' )? |
| elementpath | := | elementstep+ |
| elementstep | := | ('/' \| '//') name |
| values | := | value ( '+' value )* |
| value | := | ('.' \| '@' name \| 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 `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 120.

# Multi-path indexes

A multi-path index allows indexing multiple elements without requiring the 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 expression and a set of options. All elements matched by the XPath expression will be indexed using the given options.

While the main path is passed as an argument when creating the 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 an XPath expression used to match nodes for indexing, and a **XhiveSubPathIf** instance which carries the options for indexing the matched nodes.

The two most important configuration options for a sub-path are: **SubPathOptions.VALUE_COMPARISON** and **SubPathOptions.FULL_TEXT_SEARCH**. It is important to note that they can be used together, allowing you to search the contents of an element either through value or through full-text search. The type used for value comparison of a sub-path is String by default, but this can be changed using **XhiveSubPathIf.setType(int)**. Please consult the javadocs of **XhiveSubPathIf** for a list of all the options.

## Examples

Assuming that the main path of the index is `/main/path`, the following table describes various possibilities for specifying the XPath expression of an **XhiveSubPathIf**.

**Table 24 Sub-path XPath expression matching examples**

| Sub-path specification | Description |
|---|---|
| `/` | A single slash will match the root path of the index. In our example, this corresponds to `/main/path`. Assuming the option `SubPathOptions.FULL_TEXT_SEARCH` was used, the index will be used for queries such as `/main/path[ . contains text "some text"]` |
| `/*` | A path such `/*` matches any direct child element of the main path of the index. Assuming option `SubPathOptions.FULL_TEXT_SEARCH`, such a sub-path configuration would allow the index to speed up queries such as `/main/path[elem contains text "some text"]`, or `/main/path[elem2 contains text "some text"]`. |
| `elem` | Indexes all elements named `elem` which are direct children of the index' main path, so elements with path `/main/path/elem`. If this sub-path uses the option **SubPathOptions.VALUE_COMPARISON**, it would be used in a query such as `/main/path[elem = "value"]`. If using the option **SubPathOptions.FULL_TEXT_INDEX**, it could be queried with `/main/path[elem contains text "some text"]`. |
| `elem/elem2` | This is a trivial extension of the previous example. Elements with path `/main/path/elem/elem2` will be indexed. |
| `//elem` | This path will match any `elem` node. However each distinct element path will be indexed separately. Thus, assuming usage of option **SubPathOptions.VALUE_COMPARISON**, the index will be used for queries like `/main/path[elem = "value"]`, or `/main/path[A/B/C/elem = "value"]` but not for a query such as `/main/path[//elem = "value"]`. |
| `//*` | 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 subpath with the subpath options `FULL_TEXT_INDEX` and `INCLUDE_DESCENDANTS` might create a very large index, depending on your XML document structure. It will cause all tokens from descendant elements to be include in parent elements. See also Multipath Index Limitations, page 118. |

Users familiar with xDB's path indexes may be confused with the similarities and differences between path indexes and multi-path indexes. The following table tries to illustrate the differences between both index types:

**Table 25 Differences between multi-path indexes and path indexes**

| Aspect for comparison | Multi-path 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 | Multi-path indexes may have several different sub-paths indexed, and each sub-path is indexed independently of one another. 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 it is **not** possible to query this index through `/main[ a = "value" ]`. |

| Aspect for comparison | Multi-path index | Path index |
|---|---|---|
| Indexing attributes of elements | Multi-path indexes currently have no support for indexing attributes. | Path indexes can index attributes through the syntax `/elem[@attributeName]`. |
| full-text components | Multi-path indexes do not impose limits on the number of sub-paths that use the option `SubPathOptions.FULL_TEXT_SEARCH`. | Path indexes can only have a single element being indexed as a full-text index. Meaning that 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 | multi-path indexes allow a sub-path to specify the option **SubPathOptions.VALUE_COMPARISON** together with **SubPathOptions.FULL_TEXT_SEARCH**, and in that case it will index all elements matching the sub-path both by valud as well as full-text. | path indexes also allow you to specify a node twice once being indexed per value, and another as ful-text but as you have to search for these elements together, this is ineffective. |

### 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 multi-path indexes*

Multi-path indexes allow customizing the score of the whole index through a user defined class. This class, like the analyzer, must be set in the configuration object before the creation of the index. This can be done through **XhiveMultiPathConfigurationIf.setScoreCustomizer(String)**.

The **XhiveScoreCustomizerIf** is an interface that extends Lucene's **Similarity** methods by adding a **XhiveScoreCustomizerIf.getScoreBytes**. This class works in the same way as Lucene's **Similarity**, the main difference 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

## *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 subpaths that either specify the `INCLUDE_DESCENDANTS` option or with subpaths that index all fields below a path, e.g., `//*`.

In addition, subpaths 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 subpath `//*` 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 note that for subpaths like `//*`, this might cause a significantly larger index, as every parent element will contain all tokens of its descendant elements. Depending on the structure of your XML document, this will cause a significantly larger index.

# Value indexes

A value index stores elements by an element value or attribute value. Value indexes are created for a library or document. An index list can contain multiple value indexes.

There are three different types of value indexes:

- Value indexes that store elements by element value.
- Value indexes that store elements by attribute value.
- Value indexes that store named elements by attribute value.

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.

### Example

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 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");
```

## *Value index types*

Different 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.

All value types match XML Schema simple data types. By default a value index contains string values.

When selecting a type for the index, it is important that the indexed data adheres to the value type. If not, xDB throws an exception. 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.

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. With a sorted index, 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.

### API documentation

com.xhive.index.interfaces.XhiveIndexIf

# 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 are live indexes that are automatically updated when elements and attributes are inserted, replaced, or removed.

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. |

Specifying a custom analyzer class allows finer control over the 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 184.

Full-text indexes support namespaces. 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 code example 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);
```

# 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']`.

### API documentation

com.xhive.index.interfaces.XhiveIndexListIf

# 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']`.

### API documentation

com.xhive.index.interfaces.XhiveIndexListIf

# Library indexes

A library indexes is used to index the content of a library. Library indexes are live indexes that are updated automatically when library content is added, replaced, or removed. Library indexes must have a unique name and are stored in an XhiveIndexListIf list index.

There are two types of library indexes:

- Library ID index.
- Library name index.

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 and one library name index. Adding a second library ID or library name index generates an exception.

### Library ID index

A library ID index stores library content, such as documents, libraries, and BLOBs using the IDs of the content objects. This index improves the performance and scalability of the get(long Id) method in XhiveLibraryIf interface. Library ID indexes are efficient when many content objects are stored in the library.

The following example sample describes 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);
```

### Library name index

A library name index stores library content, such as documents, libraries, and BLOBs using the name of the content objects. This index improves the performance and scalability of the get(String name) method in the XhiveLibraryIf interface. Names are not mandatory for library content and some content is not included in the index. Library name indexes are best used with the get(String name) method. The library name index improves performance of XLink operations and full path XPointer queries.

The following example describes 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 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.

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 indexes are live indexes : they are automatically updated when element IDs are inserted, replaced, or removed. 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 example describes 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 example describes 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. 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.

To distinguish between elements that have the same local name but different namespaces, the keys must be namespace-aware. An element is indexed with the following key:

- String *nodeName*, when the namespaceURI is null.
- String *namespaceURI* + ' ' + *localName*, when the namespaceURI is not null.

  The string contains both namespaceURI and local name, divided 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**.

## Example

Element name indexes can be created for a library or document.

The following example code describes 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"};
```

# 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.

## *Creating context conditioned indexes*

Context Conditioned Indexes are deprecated as of 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 following code sample describes how to use 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());
}
```

# Concurrent indexes

Indexes can be specified as concurrent, using the XhiveIndexIf.CONCURRENT flag when creating the index. 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.

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.
- Uniqueness checking is not implemented for concurrent indexes. The extra locking required to ensure uniqueness would defeat the main purpose of concurrent indexes. Checking that a key is not present is not sufficient. If a key removal has not been committed yet, it could still be rolled back.
- 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.

# Optimizing index performance

To achieve the best index performance:

- Do not add more indexes than required.
- Reduce index scope.
- Make the index selective.
- Use selected element name indexes instead of default element name indexes.
- Create indexes after loading the data.

## 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 bigger than the scope of a nested library. The smaller the scope of the live index, the less index updates a data update triggers within the selected scope. Therefore the smaller the index scope, the better the performance of data updates and queries.

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

Certain indexes can be disabled providing a comma-separated list of index names in 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 ...
```

# Chapter 9

# Administering xDB

This chapter contains the following topics:



# Administration client

The administration client provides administrators and superusers access to key database functions. The administration client is based on the xDB API and offers the same functionality as a Java program that uses the API. The xDB installation includes the administration client source code, allowing developers to inspect how the administration client uses the xDB API to perform certain tasks.

## *Starting the administration client*

To start the xDB administration client:

1. Execute the **xdb admin** command in the *XhiveDir* `\bin` directory of the xDB installation. On Windows platforms select **xdb admin** in the start menu.

   An alternative way to run the administrator client is to execute the **xhive-ant run-admin** command in the *XhiveDir*`\bin` directory of the installation. The command compiles and runs the administrator client from the included source files.

   In xDB version 7.0 and higher, point the `JAVA_HOME` environment variable in the `xhive-ant` file to JDK 1.5.

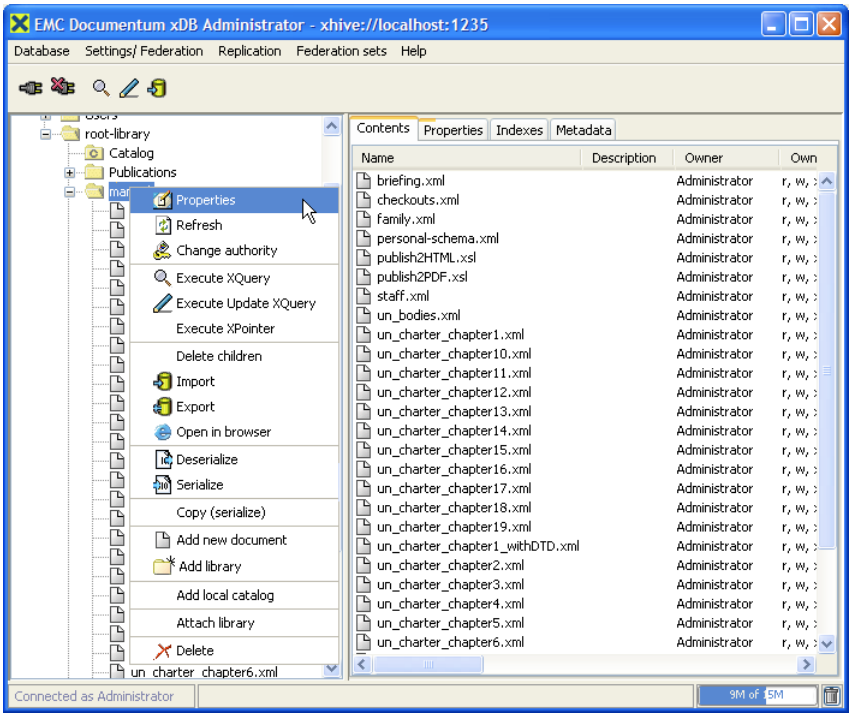
2. Select **Database > Connect**.

   The Connect to a database window displays.

3. Enter the database name, user name, and password of a valid user.

The administration client is a database explorer which contains a tree view of the database information in the left panel. The right panel displays details of the node selected in the tree view. Most operations are accessible by right clicking on nodes in the tree view.

## Figure 7 Administration client tree view




There are three main nodes in the database tree:

- Database info node

  This node provides information about segments, container pools, and cluster rules.

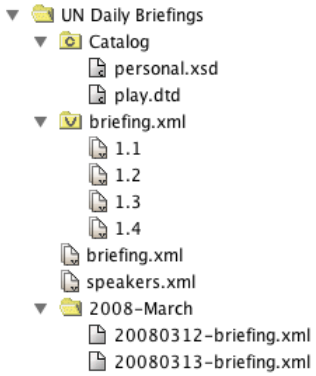
- Groups and Users node

  This node is used for viewing, adding, and removing users and groups.

- Root library node

  The root library is the top-level library that contains all other libraries and documents in a hierarchical order. Besides the libraries and documents, the hierarchy can contain special types of folders, such as catalogs and Version folders. Catalog folders are marked with the letter C and contain DTDs and XML schemas. Version folders are marked with the letter V and contain versioning information for versioned documents.

**Figure 8 Root library hierarchy**




Most tasks are accessible by right-clicking on any item in the tree view. The available tasks are displayed in a submenu. Tasks that are not accessible through the right-click menus are available in the main menu of the administration client:

- Creating and deleting databases.
- Changing the superuser password.
- Entering a new license key.

Some preferences are automatically stored, for example, the last query executed and the last database connection. These preferences are stored in the .xhive.admin.properties file in the home directory. On Windows machines, the home directory is `c:\Documents and Settings\username`. Removing the preferences file resets the administration client program.
**Note:** The status bar in the bottom right corner is a memory usage indicator. Clicking the trash can in the right corner of the administration client window forces a garbage collect.

## *Importing data*

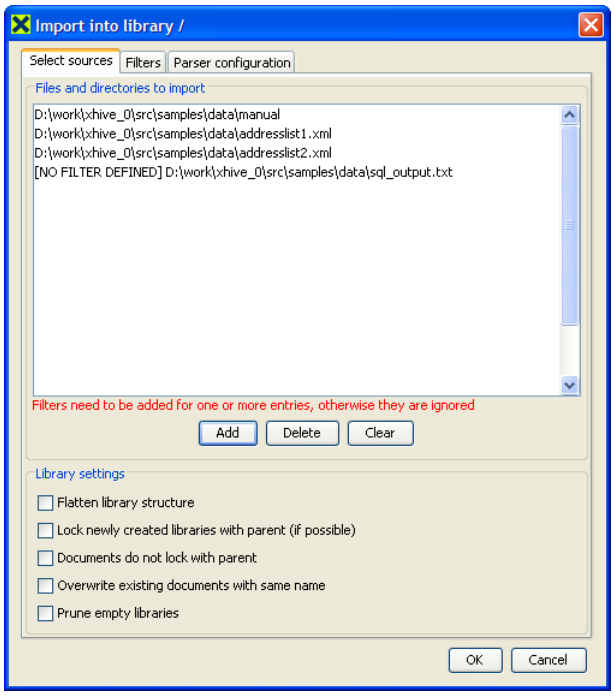
The administration client imports data using the **Import** option in the right-click menu of library nodes. The selected library is also the target for the imported data.

By default, xDB recreates the sub 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 displays with the **Select sources** tab selected.




2. Click **Add** to add files and directories to import.
3. Select the **Library settings** as follows:

| 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. |
| Documents do not lock with parents | Select this option to not lock documents with the parent. |
| Overwrite existing documents with the 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 that are imported.

   The administration client importer uses filters to determine how each file is stored. By default, the administration client importer only filters for files with .xml and .xsl extensions. Files with file extensions that are not defined, are skipped and not imported into the database. Filters are stored in the administration client preferences and are only defined once.
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. These properties can also be set on

the `DOMConfiguration` object of the `LSParser` interface. Unlike the filters, the parser configuration options are not stored in the administration client preferences.

### Related topics

DOM configuration

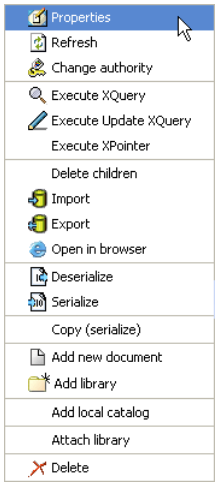
## Exporting data

Information can be exported using the **Export** menu-item in the right-click menu of library, BLOB, and document nodes.

To export data:

1. Right-click on a library, BLOB, or document node.

   The drop-down menu displays.




2. Select **Export**.
3. Select the export location and configure DOM options if required.

## Editing documents

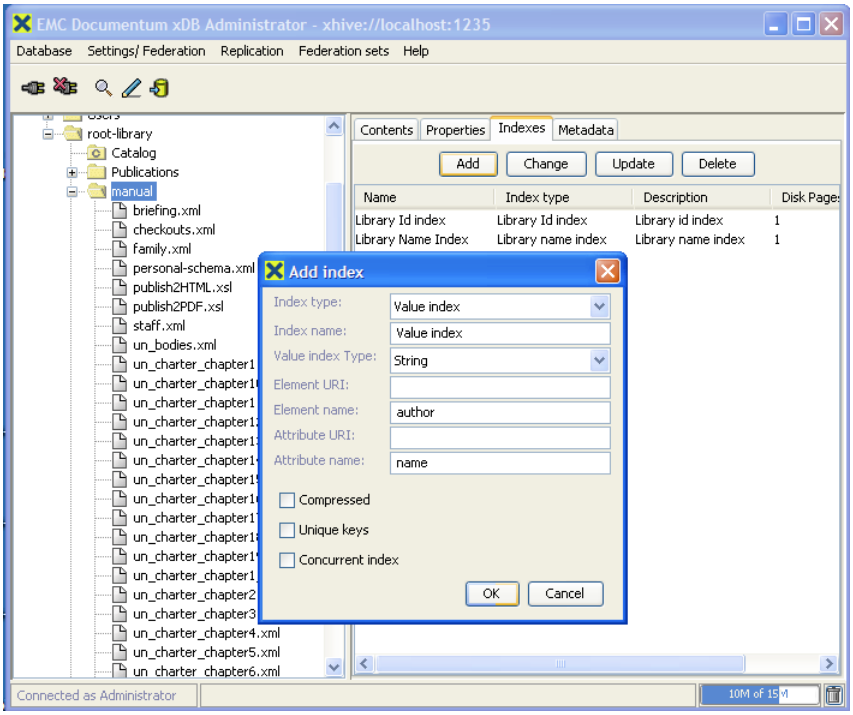
To edit documents in the database:

1. Open the administration client and browse to the document you want to edit. If the document is versioned, check out the document.
2. Right-click on the document and select one of the following options to edit the document:
   - **Browse document**Lets you browse the document as a tree and you can edit or delete individual document nodes. The Browse document option is useful for large documents.
   - **Edit as text**Lets you change the document. The Edit as text option is useful for small documents.
3. Edit the document and save your changes.

# Adding indexes

For libraries and documents the right panel of the administration client holds an **Indexes** tab for viewing and changing the list of indexes for the library. The indexes create and maintain an index for all the data contained within the sublibraries of a library.

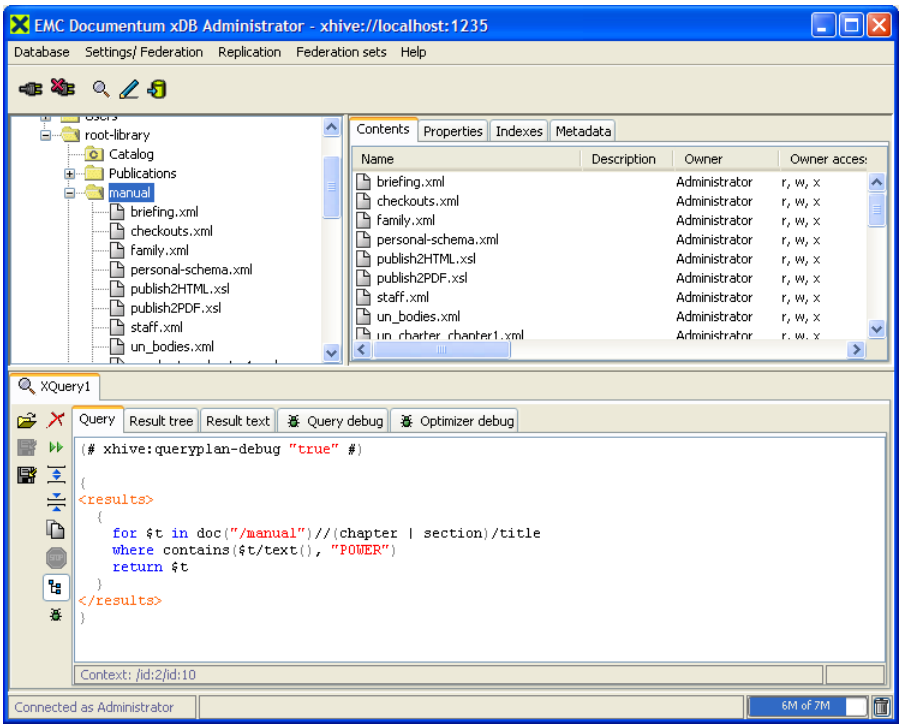
**Figure 9 Adding indexes**




# Running queries

XQuery is one of the multiple querying mechanisms supported by xDB. XPath support has been removed from the administration client because XPath is a subset of XQuery. For libraries and documents, the **Execute XQuery** option in the right-click menu opens an XQuery pane at the bottom of the screen. Queries are executed in the context of the selected the library or document.

**Figure 10 XQuery tabs and options**




The toolbar and popup menu contain two update options:

- **Execute Update XQuery**

  This option uses a read-only transaction that does not lock files to execute the query. This option can be used with the xDB XQuery update syntax, page 168.

- **Update**

  This option uses a read-write transaction that locks files to execute the query.

The XQuery panel (XQuery tab and options, page 133) contains several tabs:

**Table 28 XQuery tabs and options**

| XQuery tab | Description |
|---|---|
| Query | The Query tab lets users enter, load, debug, and save queries. The last executed query is remembered. Queries are started by clicking the >> button. |
| Result tree | The Result tree tab displays query results in a tree view. 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.<br><br>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 result can still be viewed in the **Result text** tab.<br><br>The Result Tree tab and button are disabled for update queries. |
| Result text | The Result text tab displays the query results as text. |

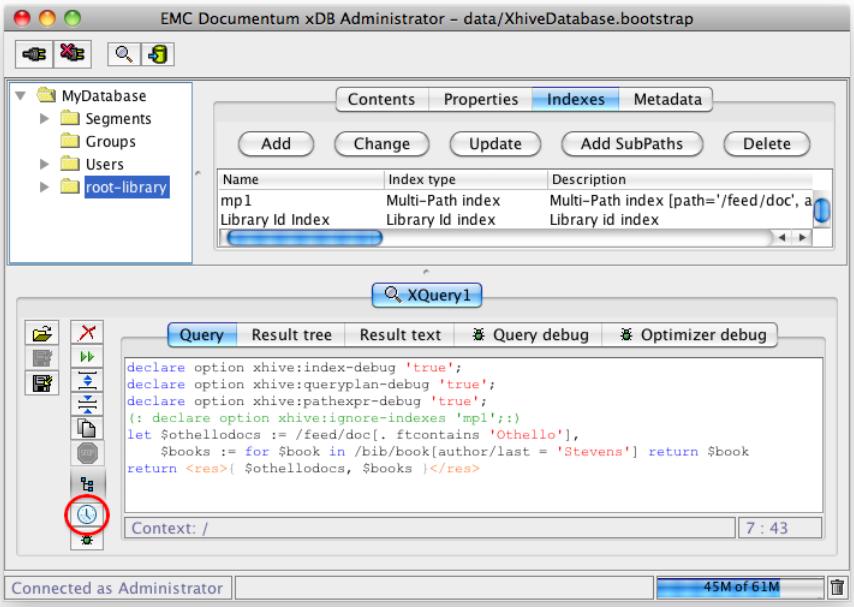
| XQuery tab | Description |
|---|---|
| Query debug | The Query debug tab displays debug information about the query, if debug options were configured in the query. For more information about debug options, see XQuery options, page 167.<br><br>The following example describes the debug output for a value index query:<br><br>`Creating query plan on node /UN Charter`<br>`for expression`<br>`  .../descendant-or-self::node()/child::chapter[@number < ...]`<br>`Looking for value index on chapter/@number (sorted) (type=INTEGER)`<br>`Found index "Value index"`<br>`Looking up "3" in index "Value index"`<br>`Using query plan: index(Value index)` |
| Optimizer debug | The Optimizer debug tag 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. |

# *Profiling XQueries*

The Administration client provides a graphical user interface to profile XQueries. This is merely a front-end to the underlying API and document format described in XQuery Profiler, page 192. That topic also contains an explanation of the document format and the meaning of attributes.

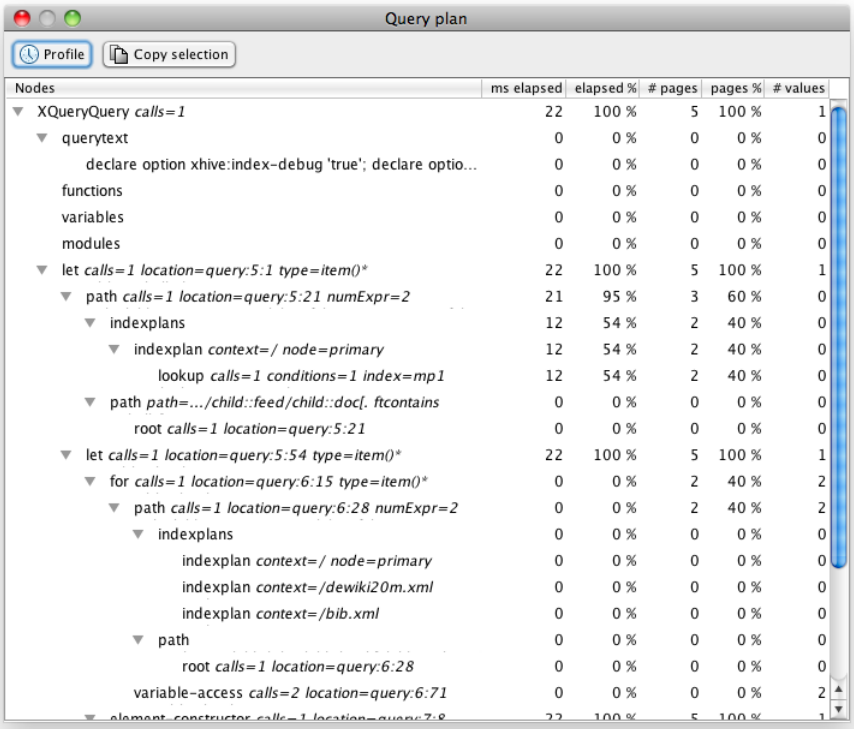
In the Administration client, after typing an XQuery, press the stop watch button highlighted below to bring up a dialog showing the static query plan.

**Figure 11 XQuery editor with profile button.**



In the query plan window, the buttons on the top allow you to run the XQuery with profiling enabled, or to copy either the complete profile (if nothing is selected) or the part of the tree rooted at the currently highlighted node.

**Figure 12 XQuery query plan dialog after profiling.**




After profiling a query, the query plan dialog shows the values of the performance-relevant attributes and a calculated percentage of the total in additional columns.

XQuery Profiler

# Command-line client

xDB includes a command-line client that allows to use and administer the database from terminals or scripting languages. The command-line client is started by executing the **xdb** command in a terminal. The xdb command is located in the bin subdirectory of the xDB installation. The Windows installer automatically adds the command to the PATH variable.

The command line supports single commands and operation, or can be used as an interactive database console. When using the command line, commands must be entered using the following syntax:

```
xdb <command> [arguments]
```

Typing `xdb help <command>` displays the available commands and options.

The interactive console is started by using the xdb command without the command option:

```
xdb
```

Within the interactive console, all regular xDB commands are accepted. Parameters containing whitespace need to be quoted using 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, it must directly follow the option, separated by whitespace, for example –federation xhive://localhost:1235. Abbreviated options can be clustered as long as only the last option takes a value, for example -vyf xhive://localhost:1235. Any global options which are passed to the xdb command are used as the default values for any subsequent commands in interactive mode.

Interactive console commands use the same syntax as the commands that are called from a terminal. The commands are always executed in auto-commit mode. All changes are made persistent after each command, before control is returned to the command line.

```
user@localhost: ~ $ xdb
xDB 10.0.0 command line client (c) 1999-2010 EMC Corporation
Type ’help’ for a list of commands and options, type ’exit’ to leave the shell.
xhive> ll
-rwxrwxrwx Administrator  8K 17.12.08 16:37 abc.xml
-rwxrwxrwx Administrator  8K 17.12.08 22:43 un_bodies.xml
-rwxrwxrwx Administrator  8K 17.12.08 22:43 unicode.xml
-rwxrwxrwx Administrator  8K 17.12.08 22:43 untyped.xml
xhive> import myfile.xml /
Parsing XML document myfile.xml ...
Stored 1 file(s).
```

The interactive console will cache username, password, and database name between single commands, thus only the first command accessing a database will ask for a password if necessary.

The Command line client commands, page 136 table describes the available commands in alphabetical order.

**Table 29 Command line client commands**

| Command | Description |
|---|---|
| add-binding | Binds a library to a specified xDB server node in addition to all the existing bindings. |
| add-node | Adds an xDB server node. |
| add-segment | Adds a segment to the database. |
| admin | Starts the admin client graphical user interface. |
| attach-library | Attaches a library back into a database from a detached state. |
| backup | Backs up a complete database. |
| backup-library | Backs up one library or multiple libraries within a database. |
| cat | Prints a file. |
| cd | Changes the current working directory of the command line session. |
| change-binding | Binds a library to a specified xDB server node. |
| configure-federation | Changes federation settings (superuser password, license key). |
| consistency-check | Run a consistency check. |

| Command | Description |
|---|---|
| create-database | Creates a database. |
| create-federation | Creates a federation. |
| create-replica | Creates a replica. |
| delete-database | Deletes a database. |
| detach-library | Detaches a library if it is detachable. |
| federation-set | Manages a federation set (create, add federations, remove federations). |
| force-detach | Unconditionally detaches a segment on which the library has been created. |
| help | Prints a help message. |
| info | Prints session information. |
| import | Imports documents into the database. |
| ll | Lists database contents. |
| ls | Lists database contents. If option "details" is specified, show all properties of a library, including ids of all the segments occupied by this library. |
| mkdir | Creates directories (always creates missing parents). |
| mv | Moves or renames files or libraries. |
| put | Creates a document in the database. |
| remove-binding | Removes a specified server node as one of the binding nodes of a library. |
| remove-node | Removes an existing xDB server node. |
| remove-segment | Deletes a segment from the database. |
| repair-input-encoding | Repairs or checks input encoding after an upgrade to xDB. |
| restore | Restores a complete database from a backup. |
| restore-library | Restores one library or multiple libraries within a database from a backup. |
| rm | Deletes files/libraries. |
| run-server | Starts an xDB server node. |
| set-library-state | Sets the state of a library. |
| show-backup-metadata | Shows metadata related to a specified backup file or multiple backup files in a specified folder. Metadata fields will be separated by a space. |
| show-segment | Shows all properties of a segment, including the segment paths. Properties will be separated by space. |
| show-unusable-indexes | Shows information regarding all unusable indexes in a specified database. |
| show-unusable-libraries | Shows full paths of all unusable libraries in specified database. |
| stop-server | Stops a running xDB server node. |
| suspend-diskwrites | Suspends or resumes disk writes. |
| update-node | Updates an existing xDB server node. |
| xquery | Executes an XQuery. |

The default values for most command options are stored in the xdb.properties configuration file in the home directory of each xDB user. These options are not specified in the command line every time a command is invoked. The default values in the configuration file can be modified specifying the corresponding parameter on the command line. For more information about the configuration settings and locations, see the Configuration files, page 44 topic.

Most commands accept the common options:

**Table 30 Command line options**

| Option | Description |
|---|---|
| -u –username ARG | The user name. xDB automatically uses the superuser or Administrator user name where needed. |
| -p –password ARG | The password of the user. |
| -f –federation ARG | The federation bootstrap path or URL. |
| -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. |
| -v –verbose | Runs in extra verbose mode. |
| -V –version | Prints version information and exits. |
| -h –help | Prints this message. |

# Server-related commands

## add-node

Adds an xDB server node.

Usage: xdb add-node [options]

| Argument | Description |
|---|---|
| –logpath ARG | Log directory for the server node to be added. Optional. If not specified, the log will be created in the default location. If specified, the directory must be accessible to both the primary and the specified non-primary nodes. |
| –port ARG | Port number of the server node to be added. |
| –host ARG | Host name of the server node to be added. |
| –nodename ARG | Name of the server node to be added. |

## 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. |

### 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. |

### update-node

Updates an existing xDB server node.

Usage: xdb update-node [options]

The arguments logpath, port, and host are all optional, but you must specify at least one of them. Otherwise there is nothing to update.

| Argument | Description |
|---|---|
| –logpath ARG | 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. |
| –port ARG | New port number of the server node to be updated. Optional. |

| Argument | Description |
|---|---|
| –host ARG | New host name of the server node to be updated. Optional. |
| –nodename ARG | Name of the server node to be updated. |

### 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-node [options]

| Argument | Description |
|---|---|
| –nodename ARG | Name of the server node to stop. |

## 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 |
|---|---|
| path | 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. |
| –nodename ARG | The name of the node to which to bind the library (read-only libraries can bind to multiple nodes). |

### add-segment

Adds a segment to the database.

Usage: xdb add-segment [options] segmentid

| Argument | Description |
|---|---|
| segmentid | The id of segment whose properties will show. Supports multiple segments, separated by space. |

| Argument | Description |
|---|---|
| –path ARG | 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. |
| –maxsize ARG | The maximum size (in bytes) that the file is allowed to grow to. Optional. 0 by default. A value of 0 means the size is unlimited. |
| –temp | If specified, create a new segment for temporary data. |
| -d ARG \| –database ARG | Database name. |
| –nodename ARG | The name of the server node to which the segment should be bound. Optional. Primary node by default. |

### attach-library

Attaches a library back into a database from a detached state.

Usage: xdb attach-library [options] segmentid target

| Argument | Description |
|---|---|
| segmentid | Segment which contains the root page of the library. |
| target | Path of target library which the attached library is appended to. |
| -d ARG \| –database ARG | Database name. |

### backup- library

Backs up one library or multiple libraries within a database.

Usage: xdb backup-library [options] path

| Argument | Description |
|---|---|
| path | Path of one library or multiple libraries to backup, separated by space. |
| –overwrite | Overwrite output file if it exists. Optional. |
| -o ARG \| –file ARG | Output file. |
| -d ARG \| –database ARG | Database name. |

### 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. |

### 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. |

### 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 segment on which the library is 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. |

### remove-segment

Deletes a segment from the database.

Usage: xdb remove-segment [options] segmentid

| Argument | Description |
|---|---|
| segmentid | The id of segment to remove. Supports multiple segments, separated by space. |
| -d ARG \| –database | Database name. |

### 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. |
| –overwrite | Overwrite data files if they exist. Optional. |

### set-library-state

Sets the state of a library.

Usage: xdb set-library-state [options] path

| Argument | Description |
|---|---|
| path | Path of a library whose state will be changed. Supports multiple paths, separated by space. |
| –searchable true\|false | Set the library to searchable or non-searchable. Optional. If not specified, the library search-state remains unchanged. |
| –readonly true\|false [–recursive] | Set the library state to read-only if true, set to read-write if false. Optional. If both are not specified, the library state remains unchanged.<br>If the –recursive option is specified, set state on the library including the descendants; if not specified, only set state on the this library. |
| –unusable | Set the library state to unusable. Optional. This option cannot be used together with the preceding two options. |

### 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 |
|---|---|
| segmentid | The id of segment whose properties will show. Supports multiple segments, separated by space. |
| –datafile | If specified, only show additional information of files in the segment. It includes full path, current file size and max file size. |
| -d ARG \| –database ARG | Database name. |

### show-unusable-libraries

Shows full paths of all unusable libraries in specified database.

Usage: xdb show-unusable-libraries [options]

| Argument | Description |
|---|---|
| -d ARG \| –database ARG | Database name. |

## General commands

### ls

Lists database contents. If option "details" is specified, show all properties of a library, including ids of all the segments occupied by this library.

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 one of document, library, or detachable (only list detachable library children). |
| –details | Print detailed properties in columns separated by tabs. |

### show-backup-metadata

Shows metadata related to a specified backup file or multiple backup files in a specified folder. Metadata fields will be separated by a space.

Usage: xdb show-backup-metadata [options] path

| Argument | Description |
|---|---|
| path | If the file denoted by the path is a normal backup file, show metadata of the backup file; if the file denoted by the path is a folder, show backup metadata of every files in the specified folder. |
| –recursive | Show backup metadata of every files in the specified folder and its subfolders recursively. Ignored when the file denoted by the path is 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 database federations on a regular basis to minimize data loss in case of a system failure.

xDB supports the following backup options:

- Online backups, page 145

  This type of backup is also called **hot** backup and is be used to back up an active federation. If any xDB code is running, regular file backup utilities cannot be used.
- Incremental backups, page 146

  This type of backups stores only the data that has been modified since the most recent full or incremental backup.
- Offline backups, page 146

  This type of back is also called a **cold** backup and is used to back up an inactive federation. If no xDB code is running, any regular file backup utility can be used.
- Snapshot backups, page 147

  In this type of backup, a low-level software or hardware method is used to take snapshots of the federation.

## *Running an online backup*

If any xDB code is running, regular file backup utilities cannot be used to back up a federation. In an active federation, the data is being modified concurrently and a backup and restore operation could create an inconsistent version of the data.

Online backups can be done using the API or the **xdb backup** command that is a simple wrapper for the API.

To run an online backup, use on of the following methods:

- Run the **xdb backup** command.
- Use the **backup()** method of the **XhiveFederationIf** interface.

### Example

The following example runs an online backup using the **backup()** method.

```
XhiveSessionIf session = XhiveDriverFactory.getDriver().createSession();
session.connect("superuser", "password", null);
XhiveFederationIf federation = session.getFederation();
FileOutputStream out = new FileOutputStream("backupfile");
federation.backup(out.getChannel(), 0);
```

### Related links

Using the xdb backup command

## 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 could still flush dirty pages from the cache to the database files during the backup. The flush could result in inconsistent data in the backup files.

### Related topics

Using the xdb backup command

## Running incremental backups

Incremental backups store only the data that has been modified since the most recent full or incremental backup. The federation 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, using one of the following methods:

- The setKeepLogFiles() method of the XhiveFederationIf interface.
- The administration client.
- Editing the bootstrap file manually and setting the keep-log-files attribute of the log element to **true**. The bootstrap file can only be edited if xDB is not running.

After the keep-log-file option has been enabled, obsolete log files are only removed when a backup is created.

To create an incremental backup, use one of the following methods:

- Run the **backup** command with the **BACKUP_INCREMENTAL** option.
- Run the **xdb backup** command with the **–incremental** flag.

**Note:** If a full backup was run previous to an incremental backup, the incremental backup is only valid relative to that full backup. There can be reasons to create a full backup without disturbing the sequence of incremental backups. Creating a standalone backup does not affect the next incremental backup. A standalone backup can be run using the backup command with the BACKUP_STANDALONE option or by running `xdb backup` with the –standalone flag. It is not possible to create an incremental backup relative to a standalone backup.

### Related topics

Using the xdb backup command

## Suspending xDB activity for snapshot backups

Federation snapshots can be created using a common software or hardware method. After creating the snapshot, simply 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.

To suspend xDB write activity, use one of the following methods:

- Use the **XhiveFederationIf.suspendDiskWrites(int options)** and **XhiveFederationIf.resumeDiskWrites** methods.
- Run the **xdb suspend-diskwrites** command.

  The xdb suspend-diskwrites command supports the following options:

| Option | Description |
|---|---|
| –flush | Flushes all dirty pages in the cache to the disk. |
| –checkpoint | Takes a lightweight checkpoint. If this option is used in conjunction with the – flush option, it creates heavyweight checkpoint. Creating a backup while disk writes are suspended, a redo recovery is not necessary after restoring the backup. This option is ignored on a replicator. Replicators cannot take independent checkpoints. |
| –sync | Flushes all files to the disk. This option is useful for a low level backup mechanism that bypasses the operating system when copying the federation files. |
| –resume | Resumes disk writes after suspension. |

## Restoring backups

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.

**CAUTION: 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.**

By default, xDB restores all files to the backup location, and does not overwrite existing files.

To restore a backup, use one of the following methods:

- Run the **xdb restore** command.
- Run the **restoreFederation()** method in the **XhiveFederationFactoryIf** interface.

### Example

The following example describes restoring a federation from a backup file through the API.

```
XhiveFederationFactoryIf federationFactory = XhiveDriverFactory.getFederationFactory();
FileInputStream in = new FileInputStream("backupfile");
federationFactory.restoreFederation(in.getChannel(), null, null);
```

### Related topics

Restoring lost data from log files

Using the xdb restore command

## *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 libaries, 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.

XDB 10.0 introduced external indexes, however when restoring files related to external indexes the PathMapper object is not used, and therefore the restore procedure may overwrite the original files if they are specified through absolute paths.

## Viewing backup metadata

When you create a backup, information about the backup is stored in a backup header, such as:

- The type of backup (federation or library)
- Time of creation
- Last backup SLN
- 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?

You can read this backup header information using the XhiveBackupInfoIf interface. 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 53 API methods for reading backup metadata**

| Methods | Description |
|---|---|
| BackupType getBackupType() | Returns the backup type. Possible types are: FEDERATION, STANDALONE, INCREMENTAL, and LIBRARY. |
| String getDescription() | 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". |
| Date getCreationTime() | Returns the time when the backup was created. |
| String getDatabase() | Returns the database name of the libraries included in the backup. This method is applicable only to library backups. |
| Collection<String> getExcludedLibraries() | 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. |
| 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. |

# 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.log file. The xhive_id.log 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

# The xdb backup command

The **xdb backup** command creates an online backup to a single backup file. If the xdb backup command is used on a running server, the server has to be configured for remote clients. The xdb backup command must use the xhive://host:port format for the remote bootstrap property. The command line options for xdb backup command, page 150 table describes the available options.

**Table 54 Command line options for xdb backup command**

| Option | Description |
|---|---|
| -o –file ARG | Specifies the output file. If no output file is specified, the out is sent to standard output. |
| –incremental | Creates an incremental backup. |
| –standalone | Creates a standalone backup. |
| –keeplogfiles | Keeps obsolete log files after completing the backup. |
| –skipsegments | Specifies segments that are not backed up. |
| –overwrite | 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 the xDB server is using. Backing up from the same JVM avoids sending all the data over a TCP connection. The backup

can be done by implementing calls to the **XhiveFederationIf.backup(..)** method on the server side, or by stopping the xDB server. The backup can be run directly on the federation by passing the –federation path/to/FederationFile.bootstrap option.

## The xdb restore command

The **xdb restore** command restores a backup from a single backup file. To restore a full backup and corresponding incremental backups, the tool can be run multiple times. The command line options for xdb restore command, page 151 table describes the available options.

**Table 55 Command line options for xdb restore command**

| Option | Description |
|---|---|
| –federation <value> | 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 xdb.properties file.<br><br>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. |
| –file <value> | The name of the input file containing the backed up data. If no input file is specified, the input is read from standard input. |

# Exporting libraries and documents

xDB exports individual libraries and documents with the metadata, such as indexes, versions and authority information. This type of export is called serialization. xDB stores the data in an internal, binary format. The process is different from an online backup, as described in the online backup and serialization comparison, page 151 table.

**Table 56 Online backup and serialization comparison**

| Online backups | Serialization |
|---|---|
| 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 the administrator 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.

There are several methods for managing detachable libraries, as described in the API methods for managing detachable libraries, page 152 table.

**Table 57 API methods for managing detachable libraries**

| Methods | Description |
|---|---|
| XhiveLibraryIf.attach() | 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. |
| XhiveLibraryIf.getState() | Identifies the state of a library. |
| XhiveLibrary.setState() | 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. |
| XhiveLibraryIf.isDetachable() | Identifies whether a library is detachable. |
| XhiveLibraryIf.getNonSearchable() | Identifies whether a library is searchable. |
| XhiveLibraryIf.setNonSearchable() | Changes a searchable library to a non-searchable library. |

## *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 running consistency checker on a library to determine if the library is consistent or not, 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 API methods for managing detachable library usability, page 152 table

**Table 58 API methods for managing detachable library usability in XhiveDatabaseIf**

| Method | Description |
|---|---|
| setLibraryUsableBySegmentId(String segId, boolean usable) | Mark the detachable library whose root page is on the specified segment as usable if *usable* is true, unusable otherwise. |

| Method | Description |
|---|---|
| setLibraryUsableByPageId(long pageId, boolean usable) | Mark the detachable library that contains the specified page as usable if *usable* is true, unusable otherwise. |
| setLibraryUsableByPath(String fullPath, boolean usable) | Mark a detachable library, which is specified by the full path, as usable if *usable* is true, unusable otherwise. |
| getAllUnusableLibraries() | Return a collection of strings representing the full paths of all unusable libraries in the specified database. |
| getAllUnusableExternalIndexes() | Return all unusable external 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.

# Backing up a library

Libraries can be backed up using the **xdb backup-library** command or the **backup()** method in the **XhiveLibraryIf** interface. There are two use cases for backing up and restoring a library:

- Disaster recovery

  This process is used when a library has been corrupted, for example after a media failure.
- Archiving

  This process is used with attaching and detaching a library. When a library is not used for a long time, the library is backed up, detached, and the library data files are removed from database.. The backup file can be saved in a less expensive storage. To access to the library, the library data files are restored from the backup and the library is attached to the database.

Only the database administrator can perform library backups. Only detachable read-only libraries can be backed up and restored. Any attempt to back up a non-detachable library, or a detachable but not read-only library causes an exception. Libraries cannot be backed up incrementally.

### Related tasks

Using the API to backup a library

Using the xdb backup-library command

## *Using the API to back up a library*

A 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 xdb backup-library command

The **xdb backup-library** command creates a backup to a single backup file or to the standard output. The server must be running when this command is used. The xdb backup-library command supports the following options:

**Table 59 Command options**

| Option | Description |
|---|---|
| library | The full path to the library to backup. |
| -o –file <value> | The name of the output file. If no output file is specified, the backup is written to standard output. |
| –overwrite | Overwrites the output file if it already exists. |

### Example

The following example uses the **xdb backup-library** command to back up a library.

```
xdb backup-library --file un_charter_backup.bak "/UN Charter"
```

# Creating a federation

When xDB is installed, a federation is created automatically. Additional federations can be created using the **XhiveFederationFactoryIf** interface, the administration client, or the **xdb create-federation** command.

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 | The log file directory. If a relative path is specified, it is interpreted relative to the bootstrap file directory. |
| –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/Fedration1.bootstrap \
  --passwd secret --pagesize 4096
```

**Note:** For performance reasons, it is best practice to keep the database log files on a different physical hard disk than the database data pages.

# Read-only federations

A federation can be changed into a read-only federation by changing the bootstrap file to a read-only file. Read-only federations are useful for distributing an xDB application with a federation on read-only media, for example a CD-ROM.

A read-only federation has the following characteristics:

- 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.
- The bootstrap file is not locked. Multiple xDB applications can be run using the same read-only federation.
- The data cannot be modified.

  Even though the data itself cannot be modified, it is possible to create temporary data using a RAM segment. A RAM segment is a special type of database segment that is kept in the database cache but never written to a file. However, database performance can suffer if a large part of the cache is used to store temporary data. A RAM segment for temporary data can be enabled using the administration client or the setTemporaryDataSegment() method, as described in the following example:

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

# Federation sets

Federation sets are a way to simplify running multiple federations in a single server, using a single page cache and TCP port. A federation set is a simple XML file containing a list of federations. The **com.xhive.federationset.interfaces** package provides methods to manipulate a federation set description file, get XhiveDriverIf objects, and start federation servers.

## *Creating federation sets*

There are several ways to create a federation set description file and add federations. Adding a federation to a federation set and creating a federation are separate unrelated actions. A federation does not have to exist to be added to a federation set.

- Using the API, for example:

  ```
  String filename = ...;
  XhiveFederationSetFactory.createFederationSet(filename);
  ```

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

- 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.

### Examples

The following example runs a separate federation server.

```
xdb run-server --federationset /path/to/federationset
 --port 1235 --cache 4096
```

The application code looks like the following:

```
XhivePageCacheIf = XhiveDriverFactory.getFederationFactory().
createPageCache(4096);
XhiveFederationSetIf fs = XhiveFederationSetFactory.
getFederationSet("xhive://hostname:1235/");
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 application code.

```
XhivePageCacheIf = XhiveDriverFactory.getFederationFactory().
createPageCache(4096);
XhiveFederationSetIf fs = XhiveFederationSetFactory.
getFederationSet("/path/to/federationset");
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.

# Using the Secure Socket Layer (SSL)

For maximum security, xDB can run over an SSL connection. However, encrypting the data has 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.
- Specify system properties for the default keystore using the **-Djavax.net.ssl.keyStore=/path/to/keystore** Java command line parameter.
- Using the API to for a key manager, specifying cipher suites, and other more specialized SSL configuration.

Starting an xDB server through the **XhiveDriverIf.startListenerThread()** API, requires passing a **java.net.ServerSocket** object. SSL can be enabled by passing a **javax.net.ssl.SSLServerSocket** object that has been created with the required options.

### Configuring the client for SSL

Configuring SSL on the client includes specifying a URL as the *xhive.bootstrap* property or as the argument in the **XhiveDriverFactory.getDriver()** method. 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.

To use a custom trust manager, the same URL is be passed to the **XhiveDriverFactory.getDriver()** method with a SocketFactory object using the **XhiveDriverIf.init(int cachePages, SocketFactory socketFactory)** call.

# Checking database consistency

This task describes how to use the API to check the consistency of a database.
To check database consistency:

1. Start a session and open a connection 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);
   ```

2. Get a handle to the database you are going to check, similar to the following:

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

3. Get a consistency checker interface, similar to the following:

   ```
   XhiveDatabaseConsistencyChekerIf checker = database.getConsistencyChecker();
   ```

4. 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);
   ```

5. Check 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());
}
```

# Chapter 10

# XQuery

This chapter contains the following topics:



## Using XQuery

In xDB, XQuery queries are executed using 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.

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 that is 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 ...
}
```

The `toString()` method displays the result on the returned values regardless of their type.

Example:

```
XhiveLibraryChildIf lc = ... ;
String query = ... ;
XhiveXQueryResultIf result = lc.executeXQuery(query);
while (result.hasNext()) {
  System.out.println(result.next().toString());
}
```

For more control over serialization, nodes should be serialized using an **LSSerializer** that can be obtained from a library using **XhiveLibraryIf.createLSSerializer()**.

If the query uses node constructors, any nodes created are created in a temporary document. If desired, these nodes can be inserted into another document using the DOM **importNode()** method.

Specifying an owner document for new nodes in the call inserts the nodes into a particular document. This method is more efficient than creating a temporary document and importing its nodes into the destination document.

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. The **result.next()** method can not be called after a modification in the same session of the searched documents or libraries, or undefined results can occur. The `xhive:force()` function or the update syntax use the query output to modify the searched documents.

# External XQuery variables and functions

XQuery provides a method for importing external values into the query scope parameters. This feature requires creating 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. The **XhiveXQueryQueryIf** object is only valid for the current database session and cannot be used 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();
...
```

The **XhiveXQueryQueryIf** interface also provides a **executeOn(XhiveNodeIf node)** method, that allows running the same query multiple times on different context items. 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 61 Java type mapping**

| Java object type | XQuery value type |
|---|---|
| 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 directly used, 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 according are converted into a special Java value. For more information, see XQuery modules, page 165.

It is also possible to supply an `Iterator` over a sequence of such objects. This method 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 can be set using the `setFunction(String, XhiveXQueryExtensionFunctionIf)` method. For more information, see the `XhiveXQueryExtensionFunctionIf` API documentation. Whether the supplied function is called depends on optimization. Also, the order of the function calls can be different from what is expected.

## 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();
...
```

There are two types of the methods for setting variables and functions. One type takes a single string argument as the name and places the variable or function in the empty namespace. The other type takes two string arguments, a namespace URI and a local name. Using a namespace URI requires declaring a prefix within the query.

In xDB it is not strictly necessary to declare external variables and functions, but it is recommended for compatibility with other XQuery implementations.

# 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 a /. 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"), (: The document "document.xml" in the root library :)
doc("/MyLibrary"), (: All documents in "MyLibrary" :)
doc("/id:10"), (: The document (or all documents 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.

In xDB, the collection() function is like the doc() function but it can also be called without any parameter.

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 executeXQuery method was called, or the document containing an initial context node. The xhive:input() method can be used 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. This behaviour can be controlled using the XQuery security policy, page 165.

Applications can control document resolution in XQuery through a custom XQuery resolver, page 163.

# XQuery Resolver

xDB's XQuery engine allows applications fine grained control over the document resolution process through an XQuery resolver.

Applications can implement the interface **XQueryResolverIf** (or better, extend **AbstractXQueryResolver**), and register their implementation with **XhiveXQueryCompilerIf.setResolver(XQueryResolverIf)** or

**XhivePreparedQueryIf.setResolver(XQueryResolverIf)**. In the latter case, module and schema imports will be resolved using the default mechanism because they happen during the construction of a prepared query, and the resolver will only be invoked for documents.

`XQueryResolverIf` allows applications to control

- Document resolving triggered by `doc()` and `collection()`
- Resolution of module imports
- Resolution of schema imports

See the example XQueryResolver.java for more information.

### Samples

XQueryCompiler.java

### API documentation

com.xhive.query.interfaces.XQueryResolverIf

com.xhive.query.interfaces.AbstractXQueryResolver

# XQuery data model

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.

# 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

# XQuery Security Features

XQuery is a complete programming language, and xDB's implementation has several features that are security sensitive:

- Access to files in arbitrary locations through the doc() and collection() functions
- Execution of arbitrary Java code through the xhive:java() function and Java module imports
- Updating XML content (only in read/write transactions)
- Execution of imported XQuery code through module imports
- Importing XML schema files

Applications can control which 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. See the example XQuerySecurity.java for more information.

In addition to that, XQuery is vulnerable to injection attacks like SQL when user input is inserted by string concatenation. Applications should always use external variables, page 160 to avoid XQuery injection attacks.

### Samples

XQueryCompiler.java

### API documentation

com.xhive.query.interfaces.XhiveXQueryPolicyIf

com.xhive.query.interfaces.DefaultXhiveXQueryPolicy

# XQuery implementation

The xDB XQuery implementation is based on the XQuery 1.0 W3C Recommendation (23 January 2007) specification. xDB implements the **Full Axis Feature** and provides the ancestor, ancestor-or-self, following, following-sibling, preceding, and preceding-sibling axes.

## *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.
- xDB allows modules to declare variables and functions outside the module namespace. Those variables and functions can only be used within the module itself.
- 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.

### 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 options and extension expressions

xDB supports a set of XQuery options. Those options can be set globally in the query prologue using the following syntax:

```
declare option QName "Value";
```

An extension expression can be set as an option for a specific part of a query. Extension expressions use the following syntax:

```
(# QName Value #)
{ expr }
```

Where the `QName` option is set for the entire inner expression. The quotes around the `Value` parameter are optional. Multiple options can be set at once by writing multiple (# #) parts before the curly braces.

The following table describes the XQuery options and extension expressions.

**Table 62 XQuery options and extension expressions**

| Option or extension expression | 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 prints a message to an output stream whenever a value is looked up in an index selected by the optimizer.<br><br>The output stream is set using the XhiveXQueryQueryIf.setDebugStream() method. By default messages are printed to System.out. |
| xhive:queryplan-debug | Checks if an index is used in a query, similar to the index-debug expression and describes 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 | Checks if an index is used in a query, similar to the index-debug expression and describes which low level expressions within the XQuery are executed, and in what order. |
| xhive:optimizer-debug | Operates similar to the xhive:queryplan-debug expression and describes 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 content of the output is currently not documented.<br><br>It is recommended to set the output stream for this option using the XhiveXQueryQueryIf.setDebugStream(QName XhiveXQueryQueryIf.XHIVE_OPTIMIZER_DEBUG, PrintWriter printWriter) method, so that the output does not mask the output from other debug options, because the output is very verbose. |
| xhive:ignore-indexes | Provides a comma-separated list of indexes that should not be used to optimize accesses. |
| 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-similarity-class | Configures a fully specified classname of the similarity that is used for score calculation in the full-text query. |

| Option or extension expression | Description |
|---|---|
| xhive:timer | Specifies a timer for the encapsulated expression. The second parameter is used as a label for the timer. The timer values can be accessed via the XhiveXQueryQueryIf.getTimings() method. |
| xhive:max-tail-recursion-depth | Specifies the maximum recursion depth for tail recursive functions. The 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 the local time zone. |
| xhive:fts-implicit-conjunction | Specifies the implicit conjunction operator for full text searches. The only valid values are **AND** and **OR**. The default implicit conjunction operator is **OR**. |

### 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()
}
```

## *XQuery updates*

xDB implements the current W3C XQuery Update Facility standard (http://www.w3.org/TR/2008/CR-xquery-update-10-20080801/). This standard is in candidate recommendation state and the specified behavior can still change.

XQuery updates are evaluated using snapshot semantics. The query is evaluated completely before the updates are applied and update effects are not visible from within the query. This type of update is less error-prone, allows lazy evaluation and out of order execution.

To ensure that all updates have been generated and applied after an execute()/executeXQuery() call, 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.

xDB also supports a proprietary update syntax. Some operations on documents and libraries can only be executed using the custom syntax. The following table describes the xDB update syntax functions. These functions are deprecated, new implementations should use the official XQuery Update Syntax.

**Table 63 xDB update functions**

| Function | Description |
|---|---|
| xhive:create-library($uri as xs:string) as empty-sequence() | Creates a library with the specified $uri location. Any parent libraries that are missing in the path are created. . |
| xhive:insert-into($where as node(), $what as item()*) as empty-sequence()<br>xhive:insert-into-as-first($where as node(), $what as item()*) as empty-sequence()<br>xhive:insert-into-as-last($where as node(), $what as item()*) as empty-sequence()<br>xhive:insert-before($where as node(), $what as item()*) as empty-sequence()<br>xhive:insert-after($where as node(), $what as item()*) as empty-sequence() | Applying any of the insert functions inserts the $what value relative to the $where value. The insert-into and insert-into-as-last elements behave identically. Atomic values within the $what value are converted into text nodes similar to element constructors. If the $where value is not a node or the empty sequence, an error is raised. |
| xhive:insert-document($uri as xs:string, $document as document-node()) as empty-sequence() | Inserts the $document value at the $uri location. If there is already a document at the $uri location an error is raised. |
| xhive:remove($nodes as node()*) as empty-sequence() | Removes the $nodes values from their parents. The xhive:delete() function is an alias to this function. |
| xhive:remove-library($uri as xs:string) as empty-sequence() | Removes the library at $uri location, including all children. |
| xhive:rename-to($what as node(), $newName as xs:QName) as empty-sequence() | Renames the specified node to the $newName value. This function raises an error if the target is not an attribute node, element node, a processing instruction, or a document node. Processing instructions can only be renamed to unqualified local names, such as QNames without a namespace URI. To construct a QName, use the standard fn:QName($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() | Removes all children from the $where value and replaces them with $newContents values. This function allows to directly move DOM nodes into a new target.<br><br>This function is similar to the xhive:delete($where/node()),xhive:insert-into($where, $newContents), xhive:move($target as node(), $sources as node()*) as empty-sequence(), and xhive:move($target as node(), $anchor as node()?, $sources as node()*) as empty-sequence() functions. By default, $sources values are inserted last into $target values. If a $anchor value is specified and not empty, the $sources values are inserted before the $anchor values. |

Using the xhive:replace-value-of($where as node(), $newContents as item()*) as empty-sequence() function has a potential performance advantage over deleting and inserting nodes. If the $where and $newContents values belong to the same document, nodes do not have to 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>
  }
)
```

## *Proprietary XQuery extensions*

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 $order_col1]` instead of `$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.

# XQuery extension functions

xDB implements some useful functions that are not part of the XQuery Working Draft. These functions are all in the `http://www.x-hive.com/2001/08/xquery-functions` namespace which is bound to the `xhive` prefix by default.

- **xhive:fts($context as node(), $query as xs:string, $options as xs:string) as xs:boolean**

  This function executes a query using the full text index. The $options argument is an optional string literal containing a semicolon-separated list of options.

  Example:

  ```
  doc("/library")//chapter[xhive:fts(title, "venice and merchant*")]
  ```

  The include-attrs option executes a query using a full text index, but also looks in the attributes of $context elements.

- **xhive:evaluate($query as xs:string) 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>
  ```

- **xhive:parse($doc-text as xs:string, $schema-hint as xs:string) as document-node()** and **xhive:parse($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()***

  This function calls a Java function. For more information, see com.xhive.query.interfaces.XhiveXQueryExtensionFunctionIf API documentation.

  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:document-name($document as document-node())**

  This function returns the name of the document that is set by the XhiveLibraryChildIf.setName(String name) method. If the passed node is not a document or a document without a name, this function returns an empty sequence.

- **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 uses the query result to modify the data, which is normally impossible because of lazy evaluation.

  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);
  }
  ```

- **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** — The date on which the version was created, 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-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.

  In the following example, the query 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: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:highlight($arg as item()*, ...) as item()***

  This function calls the extension function set on the query using the XhiveXQueryQueryIf.setHighlighter(highlighter) API. 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. Any arguments passed to the XQuery function are passed as other arguments to the highlighter function.

For example, in the query

```
for $elem in //para
where $elem contains text "Rotterdam"
return xhive:highlight($elem)
```

the highlighter function is called with two arguments, **Rotterdam** and the matching `para` element. 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.

- **xhive:created-at($uri as xs:string) as xs:dateTime**

  Returns when the document, library, or blob at **$uri** was created. If **$uri** does not exist, an error is raised (err:FODC0002).

- **xhive:last-modified($uri as xs:string) as xs:dateTime**

  Returns when the document, library, or blob at **$uri** was modified for the last time. If **$uri** does not exist, an error is raised (err:FODC0002).

- **xhive:child-documents($uri as xs:string) as document-node()***

  Returns direct document children of the library indicated by **$uri**, i.e., only documents that are located directly underneath the library, not recursive descendants like the built-in **doc** function. If **$uri** does not point to a library, or the library has no children, the empty sequence is returned. If **$uri** does not exist, an error is raised (err:FODC0002).

  Note that because in 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 don't need this function.

- **xhive:child-uris($uri as xs:string) as xs:string***

  Returns the absolute URIs of direct children of the library indicated by **$uri**. If **$uri** does not point to a library, or the library has no children, the empty sequence is returned. If **$uri** 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.

## 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.

## XQuery 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 distribution). The class `java.util.Locale` specifies which locales are supported. A list can be created using the code fragment, as described 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.

## Implementation limitations

xDB does not implement the complete XQuery 1.0 specification.

The following items are not supported:

- Collations in **order by** specifications
- XQuery Static Typing Feature
- XQuery Static Typing Extensions

The following optional XQuery features are supported:

- Full Axis Feature
- Schema Import Feature
- Module Feature

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.

# Using indexes in XQuery

xDB has several index types that can be used to speed up XQueries:

- Path indexes, page 113, Multipath Indexes, page 115, Value indexes, page 118, and element name indexes, page 123

  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 120

  Full text indexes can be used through the contains text XQuery full text, page 180 operations.

- Library name indexes and library ID indexes, page 121.

  The doc() XQuery function uses library indexes.
- Id attribute indexes, page 122

  The id() XQuery function uses document ID attributes indexes. Library ID attribute indexes are only used with the xhive:get-nodes-by-key() extension function.

## *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, as described in the following example.

```
(: 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
```

Enabling the queryplan debug statements allows verifying whether an order by query is being optimized. For example, a `//book[@year<STRING> + title<STRING>]` path value index 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
```

# Full-text queries

xDB partially implements W3C XQuery Full-Text Facility Standard available at http://www.w3.org/TR/xpath-full-text-10/.

xDB also extends XQuery with a full-text search function. This proprietary syntax is still supported in xDB 10.0, but the standard syntax proposed by W3C is preferred.

xDB supports the following W3C XQuery Full-Text features:

- Logical full-text operators, page 181

xDB supports the full text operator contains text. Earlier versions of the XQuery Full-Text specification used the keyword ftcontains. It is still supported by xDB, but deprecated.

# Full-text logic operators

xDB supports the **ftor**, **ftand**, **ftnot**, and **not in** full-text logic operators. The detailed description of these operations is available at http://www.w3.org/TR/xpath-full-text-10/#logical_ftoperators.

## Examples

The following examples describe 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 "programming" ftand "web" 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 a detailed description of wildcard options, see http://www.w3.org/TR/xpath-full-text-10/#ftwildcardoption.

## Examples

The following examples describe 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.*" 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]
```

# Anyall options

xDB supports the **any**, **any word**, **all**, **all words**, and **phrase** anyall options. For a detailed description of anyall options, see http://www.w3.org/TR/xpath-full-text-10/#ftwords.

## Examples

The following examples describe 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 detailed description of positional filters, see http://www.w3.org/TR/xpath-full-text-10/#ftposfilter.

## Examples

The following examples describe 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" ftand "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" ftand "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

(: 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 detailed description of cardinality option, see http://www.w3.org/TR/2010/CR-xpath-full-text-10-20100128/#fttimes

### Examples

The following examples describe full-text search queries with cardinality options.

```
(: 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 detailed description of XQFT score variables, seehttp://www.w3.org/TR/xpath-full-text-10/#section-score-variables.

### Examples

The following examples describe 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 options.

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 xhive:fts-similarity-class XQuery option.

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.

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.

## Using the xhive:fts function

xDB extends XQuery with a full-text search function. The full text search function can be used to search for terms within a text string. Generally, terms are words. For example, the string 'yadda yadda yadda' contains three terms, each with the value 'yadda'. In xDB terms are the basic units for full text indexing and searching. The tokenization process controlled by the analyzer class determines the term. The xDB full text search implementation is partially based on the Lucene project.

Using the full-text search function has some advantages. For example the full-text search option

- Analyzes the input string as a list of terms, instead of a list of characters.
- Can use indexes.
- Allows wildcards and prefixes.
- Allows search for exact or sloppy phrases.

The full text search function uses the following syntax:

```
xhive:fts(node(s), querystring, options)
```

The first argument of the function is one or more nodes. The second argument is a query string. If a set of nodes is provided as the first argument, the full text search is executed on all the nodes. The options argument is optional and is a string literal containing a list of options separated by semicolon. The options, page 184 table describes the available full-text search options.

**Table 64 Full-text search function options**

| Option | Description |
|---|---|
| include-attrs | Searches the attribute values of elements and descendants along with other text nodes. To use the include-attrs option with full text indexes requires using the FTI_INCLUDE_ATTRIBUTES option on the index. |
| analyze-wildcards | Sends the terms in the query to the analyzer. If the query contains wildcards, the analyzer must not remove the characters that represent the wildcards. The characters are defined in the **XhiveFtsUtilIf** interface. |
| analyzer classname | The classname of the analyzer. Changes the analyzer that is used in the query by passing an analyzer classname. |

### Example

The following examples described the syntax of full text search queries.

```
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>
```

If used without any special meaning, the +, -, !, (, ), :, ^, [, ], ", {, }, ~, *, ? characters are reserved and escaped with a backslash (\) .

There are different types of queries, such as

- Boolean queries
- Prefix searches
- Phrase searches
- Searches with wildcards
- Analyzer searches

## *Boolean queries*

A Boolean query represents a composite query that can contain sub queries of arbitrary nesting level and with composition rules such as **and**, **or**, and **not**.

Each Boolean sub query has two binary qualifiers that control how the super query is matched, as follows:

- **Prohibited**

  The query is a match only when the sub query does not match. A sub query can be marked prohibited using a **-**, **!**, and **NOT**.

- **Required**

  The query is a match only when the sub query is a match. This condition is necessary but not sufficient condition for the super query to match. Queries can be marked required using a +.

The default implicit conjunction is **OR**. A query such as "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.

XQuery has some functionality overlap. For example, the queries

```
//element[xhive:fts(., "apples AND oranges")]


(# xhive:fts-implicit-conjunction 'AND' #) {
//element[xhive:fts(., "apples oranges")]
}
```

generate the same results as the following query

```
//element[xhive:fts(., "apples") and xhive:fts(., "oranges")]
```

There is no semantic difference between the three options. However, in the current implementation the first two types of query are faster.

## 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. For example, the phrase query 'winding road' matches 'winding road' but not 'road winding', except for more relaxed slop factors.

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 .

## 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 able to search for terms with a wildcard as the first character. If this option is not set, the search can become extremely slow.

## Analyzer searches

An analyzer breaks up content in tokens and can also change tokens to improve the search capacity. For example, the analyzer can change the 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, it depends on the value of the fts-analyzer-class option 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 used by the fts function

- 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".

## *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.

# Using type information in XQuery

There are two ways in which type information is stored in xDB:

- Value indexes can have a type, page 119.
- During validation of a document, PSVI, page 82 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)]
```

### Samples

TypedIndex.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))
```

# 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.setParallelExecution(executor);
XhiveXQueryResultIf result = query.execute();
while (result.hasNext()) {
   result.next();
}
result.close()
```

While parallel evaluation can reduce the response time of queries, there is some overhead involved that can reduce the total throughput.

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 the same expressions that can use indexes, regardless of whether indexes are present or used. For more information about optimizing expression, see Value indexes, page 118. If the xhive:queryplan-debug option has been turned on for the query, the output contains a message if the query is being parallelized.

Please note that 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.

# 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 the 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 is 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` speeds up the query by allowing the evaluator to elide sorting 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 necessary improves 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.

It is possible to verify if a function is tail recursive by enabling the `declare option xhive:queryplan-debug 'stdout';` queryplan debug option. The maximum tail recursion depth can be set using the `xhive:max-tail-recursion-depth` option, as described in XQuery options, page 167.

### Samples

ParallelQuery.java

# XQuery Profiler

To find out which part of an XQuery takes a long time to execute, or which part reads a lot of data, xDB can create a profile of an XQuery execution. This can explain why a query runs slow.

**Note:** This document discusses the XML format of an XQuery profile. This format is work in progress and subject to change without notice.

xDB provides a static XML representation of an XQuery query plan through the methods **XhivePreparedQueryIf.getQueryPlan(DOMImplementation)**, **XhiveXQueryQueryIf.getQueryPlan()**, and **XhiveXQueryResultIf.getQueryPlan()**. These methods return an XML document that 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).

When an XQuery is executed with the `xhive:profile` option enabled (as an XQuery option, or through the XQuery compiler API), after at least partial execution the XQuery (by calling **XhiveXQueryResultIf.next()**), will contain profiling information.

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. `pagesRead` will only be accurate if the query is run in a "new" session immediately after a call to `begin()`.
**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 of a the parent expression relative to the variable-binding expression will typically be correct.

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.

xDB's AdminClient provides a simple graphical user interface to profile XQueries and view the profile, page 134.

## 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 'mp1';:)
let $othellodocs := /feed/doc[. contains text 'Othello'],
    $books := for $book in /bib/book[author/last = 'Stevens'] return $book
return <res>{ $othellodocs, $books }</res></querytext>
      <functions/>
      <variables/>
      <modules/>
      <let accumulatedTime="10" calls="1" location="query:5:1" pagesRead="5"
           type="item()*" values="1" variable="othellodocs@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="mp1" 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 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-constructor 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="othellodocs@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-constructor>
      </let>
   </let>
</XQueryQuery>
</queryplan>
```

Profiling XQueries using the AdminClient

# Group by in XQuery

Group by is a common operation in database queries that groups result elements that have a common key. XQuery 1.0 does not have a built in group by operator, but the same result can be achieved using a combination of the distinct-values function and for or let statements.

Given the following XML:

```
<bib>
   <book year="1992">
       <title>Advanced Programming in the Unix environment</title>
       <author>Stevens W.</author>
   </book>

   <book year="2000">
       <title>Data on the Web</title>
       <author>Abiteboul Serge</author>
   </book>

   <book year="1994">
       <title>TCP/IP Illustrated</title>
       <author>Stevens W.</author>
   </book>
</bib>
```

To retrieve books grouped by their authors, we can use the following XQuery:

```
for $author in distinct-values(doc("/bib.xml")/bib/book/author)
return
  <group>
    <author>{
      $author
    }</author>
    <books>{
      doc("/bib.xml")/bib/book[author = $author]
    }</books>
  </group>
```

Running the query will return the following XML:

```
<group>
  <author>Stevens W.</author>
  <books>
    <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <author>Stevens W.</author>
    </book>
    <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author>Stevens W.</author>
    </book>
  </books>
</group>
<group>
  <author>Abiteboul Serge</author>
  <books>
    <book year="2000">
      <title>Data on the Web</title>
      <author>Abiteboul Serge</author>
    </book>
  </books>
</group>
```

Usually, this query would traverse the data twice: once to find all the distinct autors, and once to retrieve the books written by each author. xDB detects such situations and rewrites the query so that it only traverses the XML tree once. The optimizer will print out a debug statement if the queryplan-debug is enabled, for example by prepending "declare option queryplan-debug 'true';" to the query:

```
query:12:7:Detected group by equivalent clause, rewriting query.
```

# 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 all exceptions related to XQuery, as described in the XQuery exceptions, page 196 table.

**Table 65 XQuery exceptions**

| Exception | Description |
|---|---|
| XhiveXQueryErrorException | This exception is thrown when the query contains semantic errors. |
| XhiveXQueryTypeException | This exception is thrown on 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 | This exception is thrown on stack overflows in user defined functions. This exception is a subclass of the XhiveXQueryErrorException class. |
| XhiveXQueryParseException | This exception is thrown when the query contains parse errors. |
| XhiveXQueryFTSParseException | This exception is thrown when an FTS query is incorrect. |
| XhiveXQueryUnknownFunctionException | This exception is thrown when the query uses an unknown function. |
| XhiveXQueryUnsupportedException | This exception is thrown when the query uses an unsupported feature or the declared XQuery version is greater than 1.0. |
| XhiveXQueryInternalException | This exception is thrown when the query contains internal errors. |

# Chapter 11

# Session and Transaction Management

This chapter contains the following topics:



## Sessions and transactions

xDB accesses the database using transactions within sessions. Sessions are connections to a database. A session can have multiple transactions that contain actions that are performed on the database, such as changes or rollbacks.

Whenever data is modified during a transaction, the data is locked. The locks are released when a transaction is committed or rolled back. When one transaction contains locked data, other transactions cannot modify that data at the same time.

The xDB session 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 written to the disk.

# Session lifecycle

At a minimum, a full session lifecycle consists of the following operations:

- createSession(), page 199
- join(), page 200
- connect(), page 199
- begin(), page 199
- Perform database actions
- commit(), page 199 or rollback(), page 200
- leave(), page 200
- disconnect(), page 199
- terminate(), page 200

Follow this model closely when creating xDB applications.

## Example

The following code can be downloaded from the EMC Developer Network http://community.emc.com/community/edn/documentum. The sample servlet describes how to use sessions with session pools.

```
XhiveSessionIf session = sessionPool.getSession();
try {
  session.join();
  session.begin();

  executeRequest(session);

  session.commit();
} finally {
  if (session.isOpen()) {
    session.rollback();
  }
  if (session.isJoined()) {
    session.leave();
  }
  sessionPool.returnSession(session);
}
```

The structure is simple and contains only one conditional statement. The `finally`-block ensures that the session transaction never remains open. When an exception occurs in the `try`-block, the session is rolled back, otherwise the commit command has succeeded.

In this sample the actual database operations are performed in the `executeRequest()` method, which does not contain any session-related operations at all.

## API documentation

com.xhive.core.interfaces.XhiveSessionIf

# XhiveDriverIf.createSession()

The createSession() creates a session.

### Example

The following example retrieves a session.

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

# 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 that 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.

# 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 number of changes that are made in the transaction.

# connect()

The connect() method connects a session to a database. The connect method has a relatively small overhead. It is no problem to connect and start a new session for individual users in a multi-user setting.

# 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.

## 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.

Always call the join() method before using a session in a certain thread. When working with servlets or EJBs, each request is executed in an unknown thread. During the execution the thread does not change and the join() method is called once.
**Note:** It is not possible to use a single session concurrently in multiple threads. Do not 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 can 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 is not possible to terminate the session after the thread is exited and the terminate() method is called from another thread.

## 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.

## terminate()

The terminate() method closes the TCP connection to the 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. The finalizer closes any TCP connection to the server. The sessions in open transactions have internal references and are never garbage collected. Sessions that are not part of a transaction are garbage collected when all references are released.

# 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.

# Transaction isolation in sessions

xDB supports transaction isolation and atomicity. When the begin() method is called, the database view for that session is updated and the changes made in other transactions within other sessions are visible. The following example contains three sessions and describes when a change in read-write session A becomes visible for the read-only sessions B and C. Each column describes the actions in a session in chronological order.

```
Session A           |    Session B           |     Session C
                    |                        |
begin()             |                        |
                    |                        |
                    |    begin()             |
                    |                        |
addDocument 'doc'   |                        |
                    |                        |
                    |    // 'doc' not seen   |     begin()
                    |                        |
                    |                        |     // 'doc' not seen
                    |                        |
                    |                        |     commit()
                    |                        |
commit()            |                        |
                    |                        |
                    |    // 'doc' not seen   |     begin()
                    |                        |
                    |    commit()            |     // 'doc' is seen
                    |                        |
                    |    begin()             |     commit()
                    |                        |
                    |    // 'doc' is seen    |
                    |                        |
                    |    commit()            |
```

The document added in the transaction within session A is not visible in any other transaction until the transaction within session A is committed. The open transaction of session B does see the document that was added until calling the next begin() method.

# Session locking

To ensure that the same data cannot be modified in different transactions, xDB locks the data during modification. These locks are placed on the related objects as soon as they are modified, and they are released after a commit() or rollback() call. When an object is locked, other transactions cannot change it at the same time. By default, a transaction that attempts to use a locked object blocks until the lock is released.

Besides implicit data locking during modification, libraries can be locked explicitly by calling the lock() method.

An xDB database is divided into *locking contexts*. Within a locking context, as soon as an context is changed, the entire locking context is locked. For example, a library is a locking context. When a user adds a document to that library, other concurrent sessions cannot add any other documents to that library until the commit() method is called.

The Locking behavior, page 202 table describes several scenarios.

**Table 66 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. |

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 structure 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, two options influence the namebase locking behavior :

- **XhiveLibraryIf.LOCK_WITH_PARENT**

  By default, each library is created with its own namebase. In certain cases a new library can share the namebase of the parent library, for example if there are many libraries with little content.
- **XhiveLibraryIf.DOCUMENTS_DO_NOT_LOCK_WITH_PARENT**

  By default, all documents in a library share the same namebase. For example, in a reasonably fixed set of documents, where the contents of the documents gets changed often, each document in the library gets its own private namebase.

Using more namebases improves concurrency, but adds some space and processing overhead. The default options are a sufficient compromise for most applications.

### API documentation

com.xhive.dom.interfaces.XhiveLibraryIf

# Managing locking conflicts

The transaction cannot continue, when the transaction tries to lock a database object for writing while another transaction has read the same object. The same conflict occurs when one transaction tries to read a database object that another transaction is currently writing. 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**, xDB throws an XhiveLockNotGrantedException error 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. When the wait time has passed and the other transaction still locks the database object, xDB throws an XhiveLockNotGrantedException error.

Lock exceptions occur regardless of the different wait option. 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 moment, both transactions cannot continue, because they are both waiting for each 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 restart the transaction and try the same operation again. Using read-only transactions that do not use locks, help to alleviate the number of potential locking conflicts.

# 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.

# The xdb info command

The **xdb info** command displays debug information about open transactions and their associated locks. It is a wrapper for theXhiveDriverIf.printSessionInformation() API that sends the information to standard output. The xdb info command only applies to open sessions. Closed sessions are not listed in any internal administration and are available for garbage collection if they are no longer referenced by user code.

Locks are printed as either R(ead) or W(rite) locks, with an internal ID and a short description of the locked object. 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 option is that the first object page has been removed from the cache to create space for other pages. The object is not read from disk to retrieve the name to avoid affecting the performance of current transactions.

If an application creates different sessions for different purposes, it can be useful to name the sessions to identify them. Sessions are named using the XhiveDriverIf.createSession(String name) method.

# Chapter 12

# Replicating Federations

This chapter contains the following topics:



## Replication

xDB supports lazy primary copy replication. Primary copy means that updates are performed on a single federation, also known as the primary copy or master. The updates are propagated to the copies, also known as secondary copies, replicas, or slaves. Read-only transactions and online backups can be performed on the replicas to distribute query load. The term lazy indicates that the master does not wait for a transaction to propagate to the replicas before confirming a transaction. The replicas are updated asynchronously in the background.

The replication process sends the federation log files to the replicators. The replicators read the log files and redo all updates in their data copy.

## Creating a federation replica

There are several ways to create a replica from a federation. The replicator must be given a name that is registered with the master to inform the master that a replica is added. The master preserves any required federation log files until the replica has confirmed that it received the files.

A replicator is registered using the administration client or the **registerReplicator** method of the **XhiveFederationIf** interface. After registering the replicator, create a copy of the federation using any of the following methods:

- If no server is running for the federation, copy all the federation files using a system file copy command to move them to another machine. Do not do this while a server is running for the original federation, because it corrupts the data in the replica.

- Create a full online backup, using the **xdb backup** command, the administration client or the **XhiveFederationIf.backup** API. The backup must be made after registering the replicator. Restore the backup on the desired machine.
- Create a copy of an online federation directly, using the administration client or the **XhiveFederationIf.replicateFully** API. If desired, the replicated federation can be moved to another machine. Alternatively, run the procedure on the destination machine, connecting to the original server remotely. Use the **xdb create-replica** command or the administration client to create a replica on a replicating host.

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 administration client or the **xdb create-replica** command.

# Running a replicator on a dedicated server

The **xdb run-server** command runs a dedicated xDB server. Running the server as a replicator, requires specifying the following options:

- `-master xhive://`*host*`:`*port*

  This option passes the location of the master 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.

The replicator runs as a server, duplicating all updates at the master. Clients can connect to the server for read-only queries and online backups.

# 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);
```

# Using replicators for read-only transactions

Read-only transactions can create temporary data such as new nodes in XQuery queries or old versions of versioned documents. Because the replication mechanism updates all data in normal segments, these segments cannot be used to allocate temporary data at the replica.

Using temporary data in the replica, requires creating a temporary data segment and setting the database to use that segment for temporary data. Updates to that segment for temporary data at the master are not logged, and therefore not replicated.

Alternatively, the database can use a RAM segment for temporary data. This method is only useful if there are enough cache pages to hold the temporary data of all simultaneous sessions, with some cache space to spare for other usages.

# Replicating 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. A system configuration can require using different paths on the primary and secondary copies. If a system supports symbolic links, it is easiest to use those paths to point to the desired actual directories. 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.

# Moving the master replica

Moving the role of master copy to one of the replicas is useful if the master server is taken out of service. Before turning a master copy into a replica, the master is synchronized with the replica by shutting down the master copy and specifying the replica as new master.

The master copy can be shut down using either the administration client, the **xdb stop-server** command, or the **XhiveFederationIf.shutdown(...)** API. After the old and new master have been synchronized, a regular master server can be on the new master.

### Example

The following example uses the `xdb stop-server` command to shut down a server.

```
xdb stop-server --help
xdb stop-server --federation xhive://dbserver:1235 -wait "id1, id2"
superuserpwd
xdb stop-server --federation xhive://dbserver:1235 -wait ""
superuserpwd
```

The `--wait` option is not required. By default, the `xdb run-server` process synchronizes the master with all registered replicators. Otherwise the process only synchronizes the listed replicators. Passing an empty string as the `--wait` option instructs the process not to wait for any replicators.

# 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.

# Removing a replica

A replica is removed by stopping the replication server and removing the files. After removing the files, 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 using the administration client or the XhiveFederationIf.unregisterReplicator method.

# Chapter 13

# Optimizing Performance

This chapter contains the following topics:



## Improving internal server performance

Generally, the easiest and most effective way to speed up an xDB application is to run the xDB 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 server can run for a specific federation, it depends on the application architecture whether an internal server can be used. An internal server cannot be used, if multiple users run a client application that accesses xDB directly. However, in many application architectures all database accesses are done from a single application server. In these cases, 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. If an internal xDB server is run within an application server, it is still possible to connect. Use the administration client, a data loading utility, and the `xdb backup` command to connect. In this case the main application has to contain code like the following:

```
XhiveDriverIf driver = XhiveDriverFactory.getDriver();
driver.init(cachePages);
int port = ...;
ServerSocket socket = new ServerSocket(port);
driver.startListenerThread(socket);
```

The main application has to use the filename of the bootstrap file as the xhive.bootstrap property or as an argument to the XhiveDriverFactory.getDriver() method, instructing the application to run the xDB server internally. All other applications use a xhive://*host*:*port* URL to connect to the server that runs in the application.

To use the same internal xDB server from different applications in the same application server, both 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.

# Configuring JVM and cache

There are several JVM settings that impact the performance of Java applications, including xDB. 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 generate a large performance improvement to CPU bound processes. The performance improvement requires a larger startup time 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 xDB 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 and is insufficient for real server applications. It is impossible to recommend specific numbers. The optimal settings depend on the application and data, the hardware and the other tasks that the hardware has to perform.

# Specifying the page size

Creating a federation requires specifying a page size for the database pages. Each document and BLOB uses an integral number of pages. In most cases, it is best to use the same page size as the file system uses.

If the federation is used to store numerous small documents, it is better to use a database page size that is smaller than the file system page size to save disk space. The disadvantage is that if xDB writes a database page, the operating system has to retrieve the old file system page. This method requires copying the database page into the file system page and write back the whole file system page. When a database page size equals a file system page size, retrieving the old file system page is not necessary.

Choosing a database page size that is bigger than the file system page size is not advisable, because then file-writes are no longer atomic. If the operating system crashes and a database page is only partly written to the disk, that page, and possibly the entire database, becomes inaccessible.

On Solaris operating systems, the default file system block size is 8192 bytes. On MS Windows with a default NTFS file system and on Linux operating systems, the system block size is 4096. These default block sizes can be modified. The block sizes can be verified by running the following commands:

- The **chkdsk** command on Windows 2000.
- The **fsutil fsinfo ntfsinfo** command on Windows XP. The command displays run the number of bytes per cluster.
- The **tune2fs -l /dev/device** command on Linux with the ext2/ext3 filesystem. This command displays block size of the file system.
- The **xfs_info /mountpoint** command on Linux with xfs. The **mount** command displays the mapping between devices and logical mount points.

- The **mkfs -m /dev/device** command on Solaris or HP-UX. The command displays the block size (bsize) of the file system. The **mount** command displays the mapping between devices and logical mount points.

# Using multiple disks

If possible, store the xDB log directory on a separate disk. Writing changes to the federation logs is more performance critical than modifying the database files. A committing transaction has to wait until its log records have been 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.

# File system performance

On Linux, there are some file systems for disk formatting. The xfs file system provides good throughput on large files. For typical xDB usage, the xfs filesystem offers the best performance. For more information, see the XFS FAQ.

# 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. During a power failure or similar condition, the data in the cache cannot be written to the drive platters. It is not a problem if the operating system properly requests the disk to flush its cache when an application calls the associated method. The other option is to disable the write cache on the relevant disk drives to avoid data corruption on a power failure. This option can have a significant negative effect on performance.

If the drive contains a backup battery to guarantee that confirmed writes are always written to the physical disk even on a power failure, there is no issue.

# RPC tracing

When RPC tracing is enabled, xDB logs all RPC calls to the backend server. RPC traces are useful in performance tuning and trouble-shooting, especially in a multi-node configuration.

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

xDB supports two logging modes, standard or compact. All 16 items are included in standard mode. All items except parameters and return value are included in compact mode. The default mode is standard.

RPC call trace messages can be logged in either plain text or XML format. In plain text format, items within the same trace message are delimited by space.

The following example is an 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 msecs 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 following example is 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>
```

The default RPC trace message format is plain text. The desired mode and format can be configured in the system properties, as described in the trace modes and format table.

**Table 67 Trace modes and format**

| System property | Description | Valid values | Default value |
|---|---|---|---|
| XHIVE_RPC_TRACING_FORMAT | The trace message format. | plain - Trace message is in plain text format.<br>xml - Trace message is in XML format. | The default value is **plain**. |
| XHIVE_RPC_TRACING_MODE | The trace mode. | standard<br>compact | The default value is **standard**. |

## Sending trace output to the console

To send the trace output to the console:

1. Open the %JAVA_HOME%/jre/lib/logging.properties Java logging properties file.
2. Set the logging level for the xDB RPC tracing logger to **FINEST** as follows:

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

3. Change the logging level of the ConsoleHandler to FINEST as follows:

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

4. Set the handler to **ConsoleHandler** as follows:

   ```
   Handlers = java.util.logging.ConsoleHandler
   ```

## Sending trace output to a file

To send the trace message output to a file:

1. Open the %JAVA_HOME%/jre/lib/logging.properties Java logging properties file.
2. Set the logging level for the xDB RPC tracing logger to **FINEST** as follows:

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

3. Change the logging level of the FileHandler to FINEST as follows:

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

4. Set the handler to **FileHandler** as follows:

   ```
   Handlers = java.util.logging.FileHandler
   ```

5. Configure the trace file properties.

   The com.xhive.trace.rpc Java logger and its associated logging handler record RPC trace messages in the trace file. Java logging creates the Java logger when RPC tracing is initialized. Trace file name, max size, and other parameters can be configured in the logging.properties file keys. The FileHandler configuration keys, page 216 table describes the available keys.

**Table 68 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 Java API Doc. |
| java.util.logging.FileHandler.limit | Specifies an approximate maximum amount in bytes to write to any one file. If the value is set to 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 com.xhive.trace.log.RPCSimpleFormatter when the value of XHIVE_RPC_TRACING_FORMAT is **plain**. Set this key to com.xhive.trace.log.RPCXMLFormatter when the value XHIVE_RPC_TRACING_FORMAT is **xml**. |

Please refer to http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html for details about JDK logging properties.

6. Restart the application to apply the changes the Java logging properties file.

   **Note:** Changes to system properties are dynamic.

## 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, as described in RPC tracing, page 213.
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
   ```

## Enabling RPC tracing at application level

At application (JVM) level, RPC tracing is enabled or disabled using the enableRPCTracing() and disableRPCTracing() methods of the com.xhive.trace.RPCTracer class. When RPC tracing is enabled or disabled at application level, it is not necessary to set any system properties or Java logging properties.

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

## Enabling RPC tracing at session 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 Java logging properties.

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.

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

# Chapter 14

# Ant Tasks

This chapter contains the following topics:



## xDB Ant tasks

xDB contains custom Apache Ant tasks to create and manipulate database structures. Ant is a Java-based build tool. The Ant tasks performing common tasks associated with deploying an xDB application using simple XML-based build files. xDB require Ant 1.6 or higher.

For more information about Ant, see http://ant.apache.org/.

To work with the xDB Ant tasks create a build-file for your project. For build file examples, see the `build.xml` and `anttasktests.xml` files located in the `bin` directory of the xDB distribution.

Import the xDB tasks and types, as described in the following example:

```
<path id="MyClasspath">
   <fileset dir="c:/xhive/lib">
     <include name="**/*.jar"/>
    </fileset>
</path>
<!-- This allows you to use the xDB's tasks -->
<taskdef loaderref="xhive"
         resource="com/xhive/anttasks/tasks.properties"
         classpathref="MyClasspath"/>
<!-- This allows you to use the xDB's types -->
<typedef loaderref="xhive"
         resource="com/xhive/anttasks/type.properties"
         classpathref="MyClasspath"/>
```

The `loaderref` is needed as Ant uses different loaders for every task and type definition. In this case the loaders for the tasks and types have to be the same.

## Referencing xDB Ant types

Because the same contexts used in a typical deployment build-file are likely to be used several times in different tasks, it is possible to define them in one 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">
   <path id="MyClasspath">
     <fileset dir="lib">
        <include name="xhive-ant.jar"/>
     </fileset>
     <fileset dir="c:/xhive/lib">
        <include name="**/*.jar"/>
     </fileset>
   </path>

   <!-- This allows you to use the xDB's tasks -->
   <taskdef loaderref="xhive"
            resource="com/xhive/anttasks/tasks.properties"
            classpathref="MyClasspath"/>

   <!-- This allows you to use the xDB's types -->
   <typedef loaderref="xhive"
            resource="com/xhive/anttasks/type.properties"
            classpathref="MyClasspath"/>
   <federation id="MyFederation"
               bootstrap="c:/xhive/data/XhiveDatabase.bootstrap"
               password="MySuperUserPassword"/>
   <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:

- Setting the name attribute.
- Using nested <group/> elements.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the group that is added. This attribute cannot be used in conjunction with the nested <group/> element. | Optional. |
| quiet | Specifies whether the task progress is displayed. The default value is **false**. | No. |
| failonerror | Specifies whether the task fails if the group already exists. The default value is **true**. | No. |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database that contains the group that is created. | No |
| <user/> | The users that are members of the group. | No |

### Example

The following example uses the `name` attribute to create a group.

```
<target name="addgroup-using-name">
        <addgroup databaseref="MyDatabase.ref" name="group1" />
</target>
```

Using a nested <group/> elements to create one or more groups.

```
<target name="add-moregroups">
   <addgroup databaseref="MyDatabase.ref">
       <group name="group5" />
       <group name="group6" />
   </addgroup>
</target>
```

Using nested <group/> elements with nested <user/> elements.

```
<target name="addgroupuser">
  <addgroup databaseref="test.database">
     <group name="group1" />
     <group name="group2">
        <user name="alice" password="secret">
           <group name="another_group" />
        </user>
        <user name="bob" password="secret" />
     </group>
  </addgroup>
</target>
```

<user/> elements can be nested inside the nested <group/> elements. The users are created as members of the group.

# <adduser/>

This Ant task adds one or more users to a database.

There are two mutually exclusive ways to use this task:

- Using a name attribute to add a single user.
- Using one or more nested <user/> elements to create users.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the user that is added. This attribute cannot be used in conjunction with the nested <user/> element. | Optional. |
| 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 | Specifies whether the task progress is displayed. The default value is **false**. | No. |
| failonerror | Specifies whether the task fails if the user already exists. The default value is **true**. | No. |

### Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database that contains the new users. | No. |
| <user/> | The user that is created. | No. |

### Example

The following example uses the `name` attribute to add users:

```
<target name="adduser-target">
  <adduser databaseref="MyDatabase.ref" name="jeroen" password="secret" />
</target>
```

The following example uses the 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 or hot backup of the federation. This task can only be called if the database server is running.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| file | The file that is used to store the database backup. | Yes. |
| overwrite | Specifies whether the file attribute is used to store the backup can be overwritten. The default is **false**. | No. |
| incremental | Specifies whether n incremental backup is created that includes only the log files since the last, non-standalone backup. The default is **false**. | No. |
| keeplogfiles | Specifies whether the backup procedure removes obsolete log files after the backup. The default is **false**. | No. |

| Attribute | Description | Required |
|---|---|---|
| standalone | Creates a standalone backup. A standalone backup does not interrupt the sequence of incremental backups. Incremental backups cannot be created with respect to a standalone backup. This option implies the keeplogfiles attribute. Only one backup that is **not** standalone can be created at the same time. The default is **false**. | No. |
| quiet | Specifies whether the task progress is displayed. The default is **false** | No. |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| database | The database that is backed 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 uses the **XhiveIndexAdderIf** interface to add multiple indexes in one batch operation. This Ant task can significantly improve the indexing performance. The following indexes can be wrapped in the <batchindexadder/> element:

| Index | Description |
|---|---|
| <database/> | Adds a database with the library to which the index is added. For more information, see the the nested <library/> element example. |
| <pathvalueindex/> | Adds a path value index. |
| <elementindex/> | Adds an element index. |
| <fulltextindex/> | Adds a full-text index. |
| <idattributeindex/> | Adds an ID attribute index. |
| <libraryindex/> | Adds a library index. |
| <metadatafulltextindex/> | Adds a metadata full-text index. |
| <metadatavalueindex/> | Adds a metadata value index. |
| <valueindex/> | Adds 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>
```

## <checkdatabase/>

This Ant task checks the consistency of a database.

This task uses the XhiveDatabaseConsistencyCheckerIf.checkDatabaseConsistency API call.

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database that is checked. | Yes. |
| Checkdomnodes | The option set whether the dom nodes should be checked (default is true). | No. |
| CheckIndexes | The option set whether the indexes should be checked (default is true). | No. |
| CheckAdministrationPages | The option set whether the administration pages should be checked (default is true). | No. |
| CheckSegmentPages | The option set whether the segment pages should be checked (default is true). | No. |
| CheckPageOwner | The option set whether the pages owner should be checked (default is true). | No. |

### Example

The following code example 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>
```

# <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 | The option set whether the dom nodes should be checked (default is true). | No. |
| CheckIndexes | The option set whether the indexes should be checked (default is true). | No. |
| CheckAdministrationPages | The option set whether the administration pages should be checked (default is true). | No. |
| CheckSegmentPages | The option set whether the segment pages should be checked (default is true). | No. |
| CheckPageOwner | The option set whether the pages owner should be checked (default is true). | No. |

## 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>
```

# <checkfederation/>

This Ant task checks the consistency of a federation.

This task uses the XhiveFederationConsistencyCheckerIf.checkFederationConsistency API call.

## Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <federation/> | The federation the node belongs to. | Yes. |
| Checkdomnodes | The option set whether the dom nodes should be checked (default is true). | No. |
| CheckIndexes | The option set whether the indexes should be checked (default is true). | No. |
| CheckAdministrationPages | The option set whether the administration pages should be checked (default is true). | No. |
| CheckSegmentPages | The option set whether the segment pages should be checked (default is true). | No. |
| CheckPageOwner | The option set whether the pages owner should be checked (default is true). | No. |

### Example

The following code example checks the consistency of a federation.

```
<target name="checkfederation-target">
  <checkfederation>
    <federation refid="test.federation"/>
  </checkfederation>
</target>
```

## <checknode/>

This Ant task checks the consistency of a node.

This task uses the XhiveFederationConsistencyCheckerIf.checkNodeConsistency API call.

### Parameters

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 | The option set whether the dom nodes should be checked (default is true). | No. |
| CheckIndexes | The option set whether the indexes should be checked (default is true). | No. |
| CheckAdministrationPages | The option set whether the administration pages should be checked (default is true). | No. |
| CheckSegmentPages | The option set whether the segment pages should be checked (default is true). | No. |
| CheckPageOwner | The option set whether the pages owner should be checked (default is true). | No. |

### Example

The following code example checks the consistency of a node.

```
<target name="checknode-target">
  <checknode node="nodeA">
    <federation refid="test.federation"/>
  </checknode>
</target>
```

## <closedriver/>

This Ant task closes the federation driver.

xDB Ant Tasks leaves the XhiveDriver open after running to speed up the execution of multiple xDB tasks during a single Ant session. In this case the driver would get closed when the JVM of the Ant process exits.

This restriction can be a problem when using xDB Ant tasks spawning a new process inside the same Ant process that uses the database, like deploying a servlet using the federation driver on Tomcat.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| bootstrap | The path to the federation bootstrap file. | Yes. |

### Example

The following code example 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. The copy process does not copy empty pages. Therefore the copy of the database is smaller.

If the destination for the copy of the database exists, the task raises an exception with an XhiveException.DATABASE_EXISTS condition.

**Note:** Content conditioned indexes are not copied and only their definition remains.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The database that is copied. | Yes. |
| destination | The name of the new database copy. | Yes. |
| sourcepassword | The password of the administrator of the original database. | Yes. |
| supassword | The password of the superuser. | Yes. |
| quiet | Indicates whether output about the task progress should be displayed. The default is **false**. | No. |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| federation | The federation containing the databases. | No. |

### Example

The following code example 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 exists a new database is **NOT** created and Ant continues with the next target.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the database to be created. | Yes. |
| dbapassword | The password of the administrator for the new database. | Yes. |
| configuration | The XML file containing a custom configuration file for the newly created database. | No. |

| Attribute | Description | Required |
|---|---|---|
| quiet | Indicates whether output about the task progress should be displayed. The default value is **false**. | No. |
| failonerror | Specifies whether the task fails if the database already exists. The default value is **false**. | No. |

# <createfederation/>

This Ant task creates a federation.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| bootstrap | The name of the bootstrap file. The bootstrap file can be set explicitly by inserting a <federation/> element. | Yes. |
| password | The password of the superuser. The superuser password can be set explicitly by inserting a <federation/> element. | Yes. |
| licensekey | The xDB license key. The license key can be set explicitly by inserting a <federation/> element. | Yes. |
| pagesize | The page size in bytes. The default size is 4096. | No. |
| logdir | The path to the log file directory. If this is a relative path, it is resolved relative to the directory of the bootstrap file. The default path is **log**. | No. |
| quiet | Specifies whether the task progress is displayed. The default value is **false**. | No. |

## Example

The following code example 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. The library name must be unique. If a library with the same name exists this Ant task is ignored.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the new library. | Yes. |
| documentslock | Specifies whether documents in the new library lock with the parent. The default is **true**. | No. |
| lockwithparent | Specifies whether the new library locks with its parent. The default is **false**. | No. |
| concurrentlibrary | Specifies whether the new library can be modified concurrently. The default is **false**. | No. |
| concurrentnamebase | Specifies whether the name base of the new library can be modified concurrently. The default is **false**. | No. |
| quiet | Specifies whether the task progress is displayed. The default is **false**. | No. |

### Parameters

The following parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database in which 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 if the failonerror attribute is set to **false**.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the database to be deleted. | Yes. |

| Attribute | Description | Required |
|---|---|---|
| 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 **true**. | No. |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <federation/> | The federation from which to delete the database. | No. |

### Example

The following example 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 groups.

A group can be deleted using either the name attribute or nested <group/> elements, but not both at the same time.

### 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.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the index to delete. | Yes. |
| failonerror | Specifies whether execution should stop when an index with the specified 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 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 contains a nested <database/> and 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 in the database. If the specified library does not exist Ant stops execution unless the quiet attribute is set to true or the failonerror attribute is set to false.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the library to delete. | Yes. |
| failonerror | Specifies whether execution of the Ant task should stop when the library does not exist. The default **true**. | No. |
| quiet | Specifies whether task progress is displayed. The default **false**. | No. |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database from which to delete the library. | No. |

### Example

```
<target name="DeleteMyLibrary" depends="init">
  <deletelibrary name="MyLibrary">
    <database refid="MyDatabase">
      <library path="/MyOtherLibrary"/>
    </database>
  </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 users.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of user to delete. The name attribute cannot be used in conjunction with nested <user/> elements. | Yes. |
| failonerror | Specifies whether execution of the Ant task should stop when the user does not exist. The default **true**. | No. |
| quiet | Specifies whether task progress is displayed. The default **false**. | No. |

## Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database that contains the users to delete. | No. |
| <user/> | The user to delete. If the <deleteuser/> Ant task specifies a name attribute, the <user/> parameter is ignored. | No. |

## Example

The following example uses the `name` attribute to delete users:

```
 <target name="adduser-target">
    <adduser databaseref="MyDatabase.ref" name="jeroen" failonerror="false"/>
 </target>
```

Using one or more `<user/>` elements to delete users.

```
<target name="add-many-users">
   <deleteuser databaseref="MyDatabase.ref" failonerror="true">
      <user name="alice" />
 <user name="bob" />
   </deleteuser>
</target>
```

# <deserialize/>

This Ant task deserializes a library child. The library child in the file becomes the last child of the specified library. If no 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 | Specifies whether the task output should be displayed.<br>The default is **false**. | No. |

## Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database to contain the deserialized library. | No. |

## Example

The following example deserializes the content of the MyLibrary.xhd file.

```
<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.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| file | The source file to deserialize. | Yes. |

## Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database into which to deserialize the users and groups. | No. |

### Example

The following example deserializes the contents of the MyLibrary.xhd file.

```
<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.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the value index that is added. | Yes |
| concurrent | Specifies whether to create a concurrent index. The default is **false**. | No |
| compressed | Specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default is **false**. | No |
| elements | A comma separated list of element names to index. | No |
| quiet | Specifies whether to display task progress. The default is **false**. | No |

### 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="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.<br>By default, the current directory is the export directory. | No |
| prune | Specifies whether to create directories for empty libraries.<br>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. The `name` attribute must be used with the elementName attribute, the attributeName attribute, or both. | Yes |
| elementURI | The URI of the element that is indexed. | No |
| elementName | The name of the element that is indexed. If this attribute is not specified, the attributeName attribute must be used. | No |
| attributeURI | The URI of the attribute that is indexed. | No |
| attributeName | The name of the attribute that is indexed. If this attribute is not specified, the elementName attribute must be used. | No |
| concurrent | Boolean attribute that specifies whether to create a concurrent index. The default value is **false**. | No |
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| alltext | Boolean attribute that specifies whether indexed nodes can contain children, indexing the complete text of all nodes. The default value is **false**. | No |
| analyzer | The analyzer class name. | No |
| supportphrases | Boolean attribute that specifies whether to optimize the index to perform phrase queries. The default value is **true**. This option improves the quality of scoring results. | No |
| supportscoring | Boolean attribute that specifies whether the use of scoring is supported. The default value is **true**. | No |
| lowercase | Boolean attribute that specifies whether indexed terms are converted to lower case. The default value is **true**. When this option is used, queries are not case-sensitive. | No |
| stopwords | Boolean attribute that specifies whether words are not indexed if they are from a list of standard English stopwords. The default value is **true**. | No. |

| Attribute | Description | Required |
|---|---|---|
| supportprefixwildcard | Boolean attribute that specifies whether to support efficient searches with a leading wildcard, such as in "*TERM". The default value is **false**. Using this option can increase the size of the index considerably. | No |
| includeattributes | Boolean attribute that specifies whether to index the attribute text of indexed elements. The default value is **false**. | No |
| quiet | Boolean attribute that specifies whether task progress is displayed. 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 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 ID attribute index that is added. | Yes |
| concurrent | Boolean attribute that specifies whether to create a concurrent index. The default value is **false**. | No |

| Attribute | Description | Required |
|---|---|---|
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| unique | Boolean attribute that specifies whether to use unique keys. The default value is**false**. | No |
| quiet | Boolean attribute that specifies whether the task progress is displayed. 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 to which the index is added. | No |

### Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyIDattrbuteIndex" depends="init">
   <idattributeindex name="MyIDattrbuteIndex">
     <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 library ID index that is added. | Yes |
| concurrent | Boolean attribute that specifies whether to create a concurrent index. The default value is **false**. | No |

| Attribute | Description | Required |
|---|---|---|
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| quiet | Boolean attribute that specifies whether the task progress is displayed. 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 to which the index is added. | No |

### Example

The following example contains a nested <database/> and a nested <library/> element.

```
<target name="AddMyLibaryIDindex" 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 |
|---|---|---|
| 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.<br>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. | Yes |
| key | The name of the metadata field to index. | Yes |
| concurrent | Boolean attribute that specifies whether to create a concurrent index. The default value is **false**. | No |
| compressed | Boolean attribute that specifies whether to create a compressed index. The default value is **false**. This attribute currently only works for non-concurrent indexes. | No |
| supportphrases | Boolean attribute that specifies whether to optimize the index to perform phrase queries. The default value is **true**. | No |
| supportprefixwildcard | Boolean attribute that specifies whether leading wildcard queries will be supported. The default value is **false**. | No |
| analyzer | The analyzer class name. The default value is **none**. | No |
| lowercase | Boolean attribute that specifies whether to convert the indexed terms to lower case. This option only applies if the `analyzer` is left at its default value. Queries are not case-sensitive. The default is **true**. | No |

| Attribute | Description | Required |
|---|---|---|
| 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 `analyzer` is left at its default value. The default value is **true**. | No |
| quiet | Boolean value that specifies whether task progress is displayed. The default is **false**. | No |

### 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. The default value is **false**. | No |

| Attribute | Description | Required |
|---|---|---|
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| unique | Boolean attribute that specifies whether to allow unique keys. The default value is **false**. | No |
| valuetype | Specifies the indexed key type. This attribute can have the following values: **string**, **int**, **long**, **double**, **float**, **date**, **date_time**, **time, day_time_duration**, and **year_month_duration**. The default value is **string**. | No |
| quiet | Boolean attribute that specifies whether the task progress is displayed. 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 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>
```

## <multipathindex/>

This Ant task adds a multipath index to a library child.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the index. | Yes |
| path | The main path to be indexed. All sub-paths are made relative to this one. | Yes |
| storagepath | Alternative file system path for storing the binary files for this index. This index will be stored outside of xDB's regular segments. The default value is **none**. | No |
| scorecustomizer | The class name of a implementation of **XhiveScoreCustomizerIf**. The default value is **none**. | No |
| analyzer | The analyzer class name. The default value is **none**. | No |
| lowercase | Boolean attribute that specifies whether to convert the indexed terms to lower case. The default is **true**. This option only applies if the attribute `analyzer` is not set. | No |
| stopwords | Boolean attribute that specifies whether words are not indexed if they are from a list of standard English stopwords. The default value is **true**. This option only applies if the attribute `analyzer` is not set. | No |
| quiet | Boolean value that specifies whether task progress is displayed. The default is **false**. | No |

## 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 |
| <subpath/> | Subpath configuration for this index. If no `<subpath/>` is included. A default subpath on "`//*`" with full text seach enabled will be used. | No |

### Example

The following example 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 in the target library unless the flatten attribute is set to **true**.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| usenamespaces | Boolean attribute that specifies whether to parse namespaces. The default is **true**. | No |
| validate | Boolean attribute that specifies whether to validate the files that are parsed. The default is **false**. | No |
| namequery | Specifies the string representing XQuery to select the element containing the name for the new database. | No |

| Attribute | Description | Required |
|---|---|---|
| overwrite | Specifies whether to overwrite the document if it already exists. This attribute can have the following values:<br>• **True** â€” The existing document is overwritten.<br>• **False** â€” The existing document is not overwritten and the Ant task is stopped.<br>• **Newer** â€” The existing document in the library is overwritten if the parsed document is newer.<br>The default is value is **newer**. | No |
| flatten | Boolean attribute that specifies whether to flatten the directory structure. The default is **false**. | No |
| documentslock | Boolean attribute that specifies whether documents should lock with their parent. This attribute is only relevant when the flatten attribute is set to **false**. The default value is **true**. | No |
| 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**. The default value is **false**. | No |
| quiet | Boolean attribute that specifies whether the task progress is displayed. The default value is **false**. | No |

## 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 ID attribute index that is added. | Yes |
| path | The index path. | Yes |
| concurrent | Boolean attribute that specifies whether to create a concurrent index. The default value is **false**. | No |
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| unique | Boolean attribute that specifies whether to use unique keys. The default value is **false**. | No |
| quiet | Boolean attribute that specifies whether task progress is displayed. 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 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&lt;INT&gt;]">
    <database refid="MyDatabase">
      <library path="/MyLibrary"/>
    </database>
  </pathvalueindex>
```

```
</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. The default value is **null**. 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 |
| relativepath | Specifies whether all paths in the federation are relative. The default value is **null**. 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 |
| 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 |
|---|---|---|
| <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. The default value is **null**. 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 |
| relativepath | A boolean attribute that specifies that all paths in the federation are relative. The default value is **false**. 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 |
| quiet | A boolean operand that specifies whether the task progress output is displayed. The default value is **false**. | No |

## Example

```
<target name="restore-mybackup">
   <restore file="lastBackup.db" bootstrap="MyBootstrapFile" />
</target>
```

# <serialize/>

This Ant task serializes a library child into an output file.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| file | The destination file for the serialized data. | Yes |
| quiet | Specifies whether the task progress is displayed. The default value is **false**. | No |

### Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database with the nested elements to serialize. | No |

### Example

The following example serializes the data in the MyDatabase database and writes the output to the MyLibrary.xhd file.

```
<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.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| file | The destination file for the serialized users and groups. | Yes |

### Parameters

The following optional parameter can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <database/> | The database containing the users and groups to serialize. | No |

### Example

The following example serializes the contents of the MyLibrary.xhd file.

```
<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 |

## Example

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>
```

# <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>
```

## <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 both 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. The default value is **true**. | No |
| validate | Boolean attribute that specifies whether to validate the files that are uploaded. The default value is **false**. | No |
| psvi | Boolean attribute that specifies whether to store PSVI (Post Schema Validation Infoset) after validation of XML Documents. The default value is **false**. | No |

| Attribute | Description | Required |
|---|---|---|
| flatten | Boolean attribute that specifies whether to flatten the directory structure. The default is value is **false**. | No |
| quiet | Boolean attribute that specifies whether the task progress is displayed. The default is value is **false**. | No |

### 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 addto be library. | No |

### Example

```
<upload xmlextensions="xml,xhtml">
   <fileset dir="${data.dir}">
     <include name="fbooks/*" />
   </fileset>
   <database refid="MyDatabaseRef">
     <library path="/testLib" />
   </database>
</upload>
```

## <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>
```

# <valueindex/>

This Ant task adds a value index to a library.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the value index that is added. The name attribute also requires either the elementName attribute or attributeName attribute. | 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. The default value is **false**. | No |
| compressed | Boolean attribute that specifies whether to create a compressed index. This attribute is currently only implemented for non-concurrent indexes. The default value is **false**. | No |
| unique | Boolean attribute that specifies whether to use unique keys. The default values is **false**. | No |
| 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 |
| quiet | Boolean attribute that specifies whether to display task progress. 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 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 (if neither outputfile nor outputproperty is specified). 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. | No |

### 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. | Yes |
| <query/> | The query to run, either specified as text within the <query/> 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 |
| <param/> | A parameter to a declare variable in the XQuery. Attributes namespace, name, and value. Ant properties will be expanded in the value. | No |

## Example

The following example runs a query against a library specified using a nested <database/> and a nested <library/> element. The query takes an external variable ($addressee) which is supplied with a <param/> element, which in turn uses an Ant property.

```
<target name="run-my-xquery" depends="init">
   <propert 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 XQuery task loads the query from a file and stores the result in a file.

```
<target name="run-my-xquery-file" depends="init">
   <propert 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>
```

# xDB Ant type reference

xDB supports various Ant types. See the description of the individual Ant type for more information.

### Related references


<database/>
<document/>
<federation/>
<group/>
<library/>
<user/>
<subpath/>


## <database/>

This Ant type represents a database in a federation.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | Tthe name of the database. | Yes |
| bootstrap | Path to bootstrap file. | Yes |
| user | User name to log in. | Yes |
| password | Password of the user. | Yes |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| <library/> | A library element that may be included multiple times. | No |
| <document/> | A document element that may be included 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.

### Attributes

| 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.

### Attributes

| 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 group in a database.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| name | The name of the group. | Yes |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| `<user/>` | A database user element that may be included multiple times. | No |

## <library/>

This Ant type represents a library path in a database.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| `path` | Library path | Yes |

### Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| `<document/>` | A document in the library that may be included multiple times. | No |

### Example

```
<library id="myLibrary5" path="/existingLib"/>
```

## <subpath/>

This Ant type represents a XhiveSubPathIf instance. This type is used with a <multipathindex/> task.

### Attributes

| Attribute | Description | Required |
|---|---|---|
| `xpath` | The path to the document. | Yes |

| Attribute | Description | Required |
|---|---|---|
| `type` | Specifies the indexed element type. It can have the value `string`, `int`, `long`, `double`, `float`, `date`, `date_time`, `time`, `day_time_duration`, and `year_month_duration`. | No - default is `string` |
| `fulltextindex` | Create a full text index. | No - default is `false` |
| `valuecomparison` | Whether to index the value of this node. | No - default is `false` |
| `getalltext` | Whether to include text from subnodes to the full text index. | No - default is `false` |
| `startendmarkers` | Add start and end markers to full text index. | No - default is `false` |
| `leadingwildcard` | Enable leading wildcard search in this subpath (`*pattern`). | No - default is `false` |
| `enumerateelements` | Whether to enumerate the order of repeated elements. | No - default is `false` |
| `returningcontents` | Enable returning indexed node contents directly from index. | No - default is `false` |
| `compressed` | Whether to compress part of the index. | No - default is `false` |
| `scoreboost` | Boost factor for the score of this subpath. | No - default is `1.` |

## Example

```
<subpath xpath="line"
               fulltextsearch="true"
               valuecomparison="false"
               compressed="true"
               returningcontents="true"
               getalltext="true"
               enumerateelements="true"
               startendmarkers="true"
               leadingwildcard="true"/>
```

# <user/>

This Ant type represents a user in the database.

## Attributes

| Attribute | Description | Required |
|---|---|---|
| `name` | The user name. | Yes |
| `password` | The user's password. | No |

## Parameters

The following optional parameters can be specified as nested elements:

| Parameter | Description | Required |
|---|---|---|
| `<group/>` | A group element that may be included multiple times. | No |

# Chapter 15

# Using the API for XSL Transformations

This chapter contains the following topics:



## 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 xDB implementation of these interfaces give you a starting point for using the interfaces of those packages.

### 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("XhiveXalanTransfomer:
             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("XhiveXalanTransfomer: "
                  + 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 following example describe how to use the interface. For more information, see the Xalan documentation.

```
TransformerFactory tFactory = TransformerFactory.newInstance();tFactory.
translet = 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.

# Appendix A

# Replication Tutorial

This appendix covers the following topics:



# Preparing for replication

This task describes how to set up a simple replicating system based on an existing single server application. In this example use case the specific goal of the replicating 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. A temporary segment can be created using the administration client or the API.

To create a temporary segment:

1. Open the administration client and right-click **Segments**.
2. Add a new temporary segment by entering an ID and checking the **temporary** flag.

   Creating a temporary segment using the API requires code like the following:

   ```
   // Creating a temporary segment in Java code
   session.getDatabase().createTemporaryDataSegment("tempSegmentName", null, 0);
   session.getDatabase().setTemporaryDataSegment("tempSegmentName");
   ```

3. Create the replica by running the following command on the replica host:

   ```
   xhive 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 accessible from other hosts and that the replica host has xDB installed. The `xhive create-replica` command is a wrapper around the **XhiveFederationIf.registerReplicator(...)** and **XhiveFederationIf.replicateFully(...)** API call. The calls register the replica in the master federation and copy the complete federation over to the replica host.

# Replication application code

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 that for performance 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 you only want to run read-only transactions on your replica federation only an extra call to the **XhiveDriverIf.configureReplicator(...)**object is needed. However, in this example the replica host acts as a full copy of the application, including allowing for read-write transactions. Since xDB replication only allows updating 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 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.

# Appendix B

# Trace Examples

This appendix covers the following topics:

## Trace message in XML schema format

The following trace message is an example for a trace message in XML schema format.

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

# Index

## E



## F



## G



## H



## I

## J



## L



## M



## N



## O



## P



## Q



## R

## S



## T



## U



## V



## X