**MarkLogic 6 Questions:**

1. **Which one is async, xdmp:spawn, xdmp:eval, xdmp:invoke ?**

**Ans:**

**xdmp:spawn ()>>>**

**It is async**. Once spawned cannot be rolled back. It places the module in task queue.

**Syntax:**

xdmp:spawn(

$path as xs:string,

[$vars as item()\*],

[$options as node()?]

) as item()\*

**Parameters:**

**$path:** The path, relative to the specified root, of the module to be executed.

**$vars (optional):** The external variable values for this evaluation. This can either be a sequence of map:map objects, or a sequence of even length, alternating QNames and items/variables.

Each key in the map(s) is a string representing the name of the parameter in Clark notation: "{namespaceURI}localname". The function xdmp:key-from-QName is a convenient way to generate these keys. Each entry in the map is the value of the corresponding external variable.

Alternatively, the alternating sequence should contain QName and item pairs that specify a variable name and value for an external variable.

**$options (optional):** The options node. The default value is (). The node must be in the xdmp:eval namespace. See the xdmp:eval section for a list of options.

**Required Privilege:**

*http://marklogic.com/xdmp/privileges/xdmp-spawn*

**Usage Notes:**

The xdmp:spawn function places the specified XQuery module in the task queue to be processed. The module will be evaluated when the task server has the available resources to process it. The tasks are processed in the order in which they are added to the queue.

***Once xdmp:spawn is called, it cannot be rolled back, even if the transaction from which it is called does not complete.*** ***Therefore, use carefully or preferably avoid calling xdmp:spawn from a module that is performing an update transaction.*** Once a module is spawned, its evaluation is completely **asynchronous** of the statement in which xdmp:spawn was called. Consequently, if you call xdmp:spawn from a module that is updating a document, and if the update ends up retrying (for example, if a deadlock is detected), then the entire module is re-evaluated and the xdmp:spawn call is therefore called again. This will only happen in update statements, not in query statements. For details on how transactions work in MarkLogic Server, see "Understanding Transactions in MarkLogic Server" in the Developer's Guide.

**Example:**

xdmp:spawn("module.xqy", (),

<options xmlns="xdmp:eval">

<modules>{xdmp:modules-database()}</modules>

<root>http://example.com/application/</root>

</options>)

=> Puts the module from the modules database with the

URI <http://example.com/application/module.xqy> in the task server queue.

**Example with external variable:**

for $each-uri in cts:uris((),(),cts:collection-query("http://xyz/test"))

return (

xdmp:spawn("/spawn/restructure-test-support.xqy",

(***xs:QName("each-uri"), $each-uri)*** ,

<options xmlns="xdmp:eval"><priority>normal</priority></options>

)

)

**Example:**

(:

This example uses the <result> option to use the results of a

spawned task in the query

:)

let $x := xdmp:spawn("/oneplusone.xqy", (),

<options xmlns="xdmp:eval">

**<result>{fn:true()}</result>**

</options>

)

return

($x + 2)

(:

if /oneplusone.xqy has the following body:

1 + 1

then this query returns 4

:)

**xdmp:eval()>>**

Returns the result of evaluating a string as an XQuery module.

**Syntax:**

xdmp:eval(

$xquery as xs:string,

[$vars as item()\*],

[$options as node()?]

) as item()\*

**Parameters:**

**$xquery :** The XQuery string to be evaluated. If the XQuery string contains double quotes ("), surround the string with single quotes (').

**$vars (optional)**: The external variable values for this evaluation. This can either be a sequence of map:map objects, or a sequence of even length, alternating QNames and items.

Each key in the map(s) is a string representing the name of the parameter in Clark notation: "{namespaceURI}localname". The function xdmp:key-from-QName is a convenient way to generate these keys. Each entry in the map is the value of the corresponding external variable.

Alternatively, the alternating sequence should contain QName and item pairs that specify a variable name and value for an external variable.

**$options (optional)**: The options node. The default value is (). The node for the xdmp:eval options must be in the xdmp:eval namespace. The following is a sample options node:

<options xmlns="xdmp:eval">

<isolation>different-transaction</isolation>

<prevent-deadlocks>true</prevent-deadlocks>

</options>

**Example:**

xdmp:eval("1+1")

=> 2

**Example:**

xquery version "1.0-ml";

declare namespace my='http://mycompany.com/test';

let $s :=

"xquery version '1.0-ml';

declare namespace my='http://mycompany.com/test';

declare variable $my:x as xs:string external;

concat('hello ', $my:x)"

return

(: evaluate the query string $s using the variables

supplied as the second parameter to xdmp:eval :)

xdmp:eval($s, (xs:QName("my:x"), "world"))

=> hello world

**Example:**

xdmp:eval("doc('/docs/mydoc.xml')", (),

<options xmlns="xdmp:eval">

<database>{xdmp:database("otherdb")}</database>

</options>)

=> The '/docs/mydoc.xml' document from the otherdb database.

**Example:**

xdmp:eval('xdmp:get-current-user()', (),

<options xmlns="xdmp:eval">

<user-id>{xdmp:user("someuser")}</user-id>

</options>)

(:

returns "someuser", assuming "someuser" exists in the

security database and the user running the eval request has the

xdmp:login privilege.

:)

**Example with diff. options:**

*xdmp:eval("xdmp:node-replace(cts:search(/bookstore/book, cts:word-query('Java'))/author,<author>{'Abhinav Kumar'}</author>)",(),*

*<options xmlns="xdmp:eval">*

*<database>{xdmp:database("alfresco-pub")}</database>*

*<modules>{xdmp:modules-database()}</modules>*

*<root>/</root>*

*<isolation>different-transaction</isolation>*

*<prevent-deadlocks>true</prevent-deadlocks>*

*<default-xquery-version>{xdmp:xquery-version()}</default-xquery-version>*

*</options>*

*)*

**Xdmp:invoke()>>**

Returns the result of evaluating a module at the given path.

**Syntax:**

xdmp:invoke(

$path as xs:string,

[$vars as item()\*],

[$options as node()?]

) as item()\*

**Required Privilege:**

http://marklogic.com/xdmp/privileges/xdmp-invoke

**Example:**

xdmp:invoke("http://example.com/modules/foo.xqy")

=> 2

**Example:**

This example invokes a module using external variables.

Assume you have a module in the modules database with a URI "http://example.com/application/module.xqy" containing the following code:

xquery version "1.0-ml";

declare namespace my="my-namespace-uri";

declare variable $my:var as xs:string external;

xdmp:log($my:var)

Then you can call this module using xdmp:invoke as follows:

xquery version "1.0-ml";

declare namespace my="my-namespace-uri";

xdmp:invoke("module.xqy",

(xs:QName("my:var"), "log this to ErrorLog.txt"),

<options xmlns="xdmp:eval">

<modules>{xdmp:modules-database()}</modules>

<root>http://example.com/application/</root>

</options>)

=> Invokes an XQuery module from the modules database

with the URI http://example.com/application/module.xqy.

The invoked module will then be executed, logging the

message sent in the external variable to the log file.

**The xdmp:eval/xdmp:spawn/xdmp:invoke options include:**

**<database>**

The database ID, from xdmp:database("db\_name"), xdmp:security-database(), or xdmp:schema-database(). To specify a database other than the context database, the http://marklogic.com/xdmp/privileges/xdmp-eval-in privilege is required.

**<modules>**

The modules database ID for processing module imports. Specifying no <modules> element in the options node specifies the current modules database. Specifying 0 specifies using the file system to process module imports. You must have the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change (for xdmp:eval), http://marklogic.com/xdmp/privileges/xdmp-invoke-modules-change (for xdmp:invoke), or the http://marklogic.com/xdmp/privileges/xdmp-spawn-modules-change (for xdmp:spawn) privilege to change either the modules database to another database or to change the root to another path. You must have the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change-file (for xdmp:eval), http://marklogic.com/xdmp/privileges/xdmp-invoke-modules-change-file (for xdmp:invoke), or the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change-file (for xdmp:spawn) privilege to change the modules database to the file system or to change a root on the filesystem.

**<root>**

The root path for modules. Specifying no <root> element in the options node specifies the current root. You need the same privileges to modify the root as you need to change the module option.

**<timestamp>**

The system timestamp to use for this evaluation. Specifying no <timestamp> element in the options node specifies the most recent timestamp. You may only specify a timestamp for a query statement, not for an update statement. The timestamp is a number that is incremented by 1 each time any configuration or content change is made to the system. Specifying a timestamp of 0 uses the current system timestamp (the value returned by xdmp:request-timestamp()). Specifying a timestamp requires the xdmp:timestamp execute privilege.

**<ignore-amps>**

(Only valid with xdmp:eval, xdmp:invoke, xdmp:xslt-eval, xdmp:xslt-invoke; does not apply to xdmp:spawn, dbg:eval or dbg:invoke) Either true or false. When set to true, the statement is evaluated without using any Amps from the caller. The default value for the ignore-amps option is false.

**<isolation>**

(***Only valid with xdmp:eval or xdmp:invoke; does not apply to xdmp:spawn.)*** Either ***same-statement*** or ***different-transaction***. When set to same-statement, the statement is evaluated in the same transaction as the one from which it is called, and subsequent expressions in the calling statement will not see any updates performed in the eval/invoke/spawn. ***You can only use same-statement isolation with update statements; query statements with same-statement isolation will throw an exception.*** When set to different-transaction, the statement is evaluated in a separate transaction from the one in which it is called, making those updates available to subsequent expressions in the calling statement (assuming the calling statement is an update statement; if the calling statement is not an update, then subsequent expressions will see the version of the database at the system timestamp when the calling statement begins its evaluation). ***When using different-transaction in an update statement that calls another update statement, do not update the same document as the calling statement is updating; doing so can cause a deadlock.*** ***You cannot evaluate a statement in a different database with the isolation option set to same-statement***. ***The default value for the isolation option is different-transaction.*** For more details, see the "Understanding Transactions in MarkLogic Server" chapter of the Developer's Guide.

**<static-check>**

Specify true to only perform static checks on the module, and not to actually execute it.

**<prevent-deadlocks>**

***(Only valid with xdmp:eval or xdmp:invoke; does not apply to xdmp:spawn.)*** Specify true for the server to disallow update requests from an update transaction. **Only has an effect when the isolation option is set to different-transaction as there is no possibility of a deadlock if the isolation option is set to same-statement.** ***When set to true in an update request calling another update request, MarkLogic Server throws the XDMP-PREVENTDEADLOCKS exception.*** Setting this option to true prevents the possibility of deadlocks occurring when running eval/invoke of an update transaction from another update transaction. ***The default value for the prevent-deadlocks option is false.***

**<default-xquery-version>**

The default XQuery language version to use for the query, if the query does not contain an explicit version declaration. If this option is not provided, the defaults are:

xdmp:eval: The XQuery language version of the module that called eval. This version may vary module-by-module if a query consists of modules written in multiple language versions. It may also vary from run to run if the app-server default is changed.

xdmp:invoke: The default XQuery version for the app server that the invocation occurs on. Note that this may be different than the XQuery version of the module that calls xdmp:invoke.

xdmp:spawn: The XQuery version default set on the app server that called xdmp:spawn. The Task Server has no default XQuery version, the version to use is passed as a part of the task request.

Allowable values for this option are "0.9-ml", "1.0-ml", "1.0" and the special value "app-server". The first three are XQuery language versions. The last indicates that the default XQuery language version set on this app-server should be used. This is useful if code written in an older XQuery version needs to call xdmp:eval on strings that may have been passed as parameters, but should be interpreted in the app-server's default language version. A module may discover its own XQuery language version with xdmp:xquery-version.

**<time-limit>**

Override the default time limit with this time limit, in seconds, for this evaluation. You can set the value up to the maximum-time-limit value for the App Server in which the request is evaluated or to a lower value than the default time limit. ***This option only applies to xdmp:spawn, not to xdmp:invoke or xdmp:eval.***

**<user-id>**

Specifies the user ID for the user to run the request being evaluated (the request specified in the $xquery parameter). If no user-id is specified, then the request is run as the current user. You need to have the xdmp:login (http://marklogic.com/xdmp/privileges/xdmp-login) privilege to use the user-id option. Be aware that this is a very privileged operation, as it allows a user with this privilege to evaluate requests as any other user. For an example, ***see the fourth example xdmp:eval.***

**<default-collation>**

Specifies the collation to use for this context, unless a collation is explicitly specified in the XQuery prolog or in a function call that allows you to specify a collation.

**<priority>**

Specify the priority of the spawned task. Allowable values for this option are "***normal***" and "***higher***". **This option only applies to xdmp:spawn, not to xdmp:invoke or xdmp:eval.**

**<result>**

Return a value future for the result of the spawned task. This value future can bound be to a variable without waiting so that work can proceed concurrently with the spawned task. When the calling request uses the value future in any operation, it will automatically wait for the spawned task to complete and it will use the result. ***This option only applies to xdmp:spawn, not to xdmp:invoke or xdmp:eval. For an example, see The second xdmp:spawn example.***

**<transaction-mode>**

Explicitly set the transaction mode for this context. Valid values are **auto, query, and update**. An xdmp:transaction-mode prolog option in the evaluated query will override any transaction mode specified with this option. I.e. if this is defined in module

(***declare option xdmp:transaction-mode "update";)*** , then the value passed via options will be overridden by the module’s declaration.

**Required Privilege:**

http://marklogic.com/xdmp/privileges/xdmp-eval

You must have the http://marklogic.com/xdmp/privileges/xdmp-eval-in privilege to specify the <database> option with a database other than the context database.

You must have the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change (for xdmp:eval), http://marklogic.com/xdmp/privileges/xdmp-invoke-modules-change (for xdmp:invoke), or the http://marklogic.com/xdmp/privileges/xdmp-spawn-modules-change (for xdmp:spawn) privilege to change either the modules database to another database or to change the root to another path. You must have the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change-file (for xdmp:eval), http://marklogic.com/xdmp/privileges/xdmp-invoke-modules-change-file (for xdmp:invoke), or the http://marklogic.com/xdmp/privileges/xdmp-eval-modules-change-file (for xdmp:spawn) privilege to change the modules database to the file system or to change a root on the filesystem.

You must have the http://marklogic.com/xdmp/privileges/xdmp-login privilege to use the user-login option.

1. **xdmp:collection-delete("Test"), what will happen? if "Test" not exist**

**Ans:**

xdmp:collection-delete () Returns empty sequence always.

1. **xdmp:document-delete("test.xml"), what will happen? if "test.xml" not exist**

**Ans:**

It will throw XDMP-DOCNOTFOUND: xdmp:document-delete("test.xml") -- Document not found exception.

1. **xdmp:plan()**

**Ans:**

Returns XML element recording information about how the given expression will be processed by the index. The information is a structured representation of the information provided in the error log when query trace is enabled i.e. ***xdmp:query-trace(fn:true())***. The query will be processed up to the point of getting an estimate of the number of fragments returned by the index.

**Syntax (xdmp:plan)**:

|  |  |
| --- | --- |
| **xdmp:plan**( | |
|  | $expression as item()\*, |
|  | [$maximum as xs:double?] |
| )  as   element() | |

**Example**:

*xdmp:plan(cts:search(/bookstore/book, cts:element-attribute-value-query(xs:QName("book"),xs:QName("category"),"bk101")))*

*xdmp:query-trace(fn:true()),* **>>> This will log the below result in ErrorLog.txt file**

xdmp:plan(cts:search(/bookstore/book,cts:element-attribute-value-query(xs:QName("book"),xs:QName("category"),"bk101")))

**Result**:

*<qry:query-plan xmlns:qry="http://marklogic.com/cts/query">*

*<qry:info-trace>xdmp:eval("xquery version &amp;quot;1.0-ml&amp;quot;;&amp;#10;xdmp:query-trace(fn:true(...", (), &lt;options xmlns="xdmp:eval"&gt;&lt;database&gt;9222850557096475764&lt;/database&gt;&lt;modules&gt;791799356542601...&lt;/options&gt;)</qry:info-trace>*

*<qry:info-trace>Analyzing path for search: fn:collection()/bookstore/book</qry:info-trace>*

*<qry:info-trace>Step 1 is searchable: fn:collection()</qry:info-trace>*

*<qry:info-trace>Step 2 is searchable: bookstore</qry:info-trace>*

*<qry:info-trace>Step 3 is searchable: book</qry:info-trace>*

*<qry:info-trace>Path is fully searchable.</qry:info-trace>*

*<qry:info-trace>Gathering constraints.</qry:info-trace>*

*<qry:info-trace>Step 2 contributed 1 constraint: bookstore</qry:info-trace>*

*<qry:partial-plan>*

*<qry:term-query weight="0">*

*<qry:key>2037373649763547961</qry:key>*

*</qry:term-query>*

*</qry:partial-plan>*

*<qry:info-trace>Step 3 contributed 1 constraint: book</qry:info-trace>*

*<qry:partial-plan>*

*<qry:or-two-queries>*

*<qry:term-query weight="0">*

*<qry:key>16676395088234381515</qry:key>*

*</qry:term-query>*

*<qry:term-query weight="0">*

*<qry:key>16958302980059181939</qry:key>*

*</qry:term-query>*

*</qry:or-two-queries>*

*</qry:partial-plan>*

*<qry:info-trace>Search query contributed 1 constraint: cts:element-attribute-value-query(fn:QName("", "book"), fn:QName("", "category"), "bk101", ("lang=en"), 1)</qry:info-trace>*

*<qry:partial-plan>*

*<qry:term-query weight="1">*

*<qry:key>14365652937303545272</qry:key>*

*</qry:term-query>*

*</qry:partial-plan>*

*<qry:info-trace>Executing search.</qry:info-trace>*

*<qry:final-plan>*

*<qry:and-query>*

*<qry:term-query weight="0">*

*<qry:key>2037373649763547961</qry:key>*

*</qry:term-query>*

*<qry:or-two-queries>*

*<qry:term-query weight="0">*

*<qry:key>16676395088234381515</qry:key>*

*</qry:term-query>*

*<qry:term-query weight="0">*

*<qry:key>16958302980059181939</qry:key>*

*</qry:term-query>*

*</qry:or-two-queries>*

*<qry:term-query weight="1">*

*<qry:key>14365652937303545272</qry:key>*

*</qry:term-query>*

*</qry:and-query>*

*</qry:final-plan>*

*<qry:info-trace>Selected 1 fragment to filter</qry:info-trace>*

*<qry:result estimate="1"/>*

*</qry:query-plan>*

1. **cts:walk() vs. cts:highlight()?**

**Ans**:

The function [cts:walk](https://docs.marklogic.com/6.0/cts:walk) is similar to [cts:highlight](https://docs.marklogic.com/6.0/cts:highlight), but instead of returning a copy of the node passed in with the specified changes, it returns only the expression evaluations for the text node matches specified in the [cts:walk](https://docs.marklogic.com/6.0/cts:walk) call. Because [cts:walk](https://docs.marklogic.com/6.0/cts:walk) does not construct a copy of the node, it is faster than [cts:highlight](https://docs.marklogic.com/6.0/cts:highlight). In cases where you only need to return the expression evaluations, [cts:walk](https://docs.marklogic.com/6.0/cts:walk) will be more efficient than [cts:highlight](https://docs.marklogic.com/6.0/cts:highlight).

**Syntax (cts:walk)**:

|  |  |
| --- | --- |
| **cts:walk**( | |
|  | $node as node(), |
|  | $query as cts:query, |
|  | $expr as item()\* |
| )  as   item()\* | |

**Example**:

*cts:walk(cts:search(/pce:IC, cts:element-range-query(xs:QName("dc:identifier"),"=",13669626843246594231)),"13669626843246594231",$cts:node)*

**Syntax (cts:highlight)**:

|  |  |
| --- | --- |
| **cts:highlight**( | |
|  | $node as node(), |
|  | $query as cts:query, |
|  | $expr as item()\* |
| )  as   node() | |

**Example**:

*cts:highlight(cts:search(/pce:IC, cts:element-range-query(xs:QName("dc:identifier"),"=",13669626843246594231)),"13669626843246594231",<h>{$cts:text}</h>)*

1. **What Shallow % and deep % shows?**

**Ans:**

**Shallow % --- *Percentage time spent in sub expression***

**Deep %--- Percentage *time spent in expression and sub expression***

In XQuery, everything is an expression: values, sequences of values, functions, function calls, etc. An XQuery "program" is a tree of expressions, where some expressions are composed of other expressions. Here is an example of a main module which is a FLWOR expression:

for $book := /book[@status = 'pre pub']

let $title := $book/metadata/title

let $authors := $book/metadata/author

let $author-list := format-author-list ($authors)

return book-summary ($title, $author-list, $book/abstract)

This FLWOR is made up of several sub-expressions: the XPath ' /book[@status = 'pre pub']', the function call 'format-author-list()', etc. Each sub-expression may be further composed of an expression tree.

The profiler is hooked into the low-level expression evaluator in MarkLogic. It records time information when each expression begins evaluation and when the evaluation completes. As a given expression is evaluated, there is time spent working on that expression (shallow time) and time spent waiting for sub-expressions to be evaluated.

**Deep time:** The "wall clock" time elapsed between when an expression begins evaluation and the time it finishes.

**Shallow time:** Time spent evaluating an expression when not waiting for sub-expressions.

There are a few special cases in the numbers that I remember (things may have changed since I wrote the profiler several years ago). For example, recursive calls are not measured as sub-expressions (but they are counted), they accumulate as a single shallow time. And calls to built-in functions never have a shallow/deep breakdown because they are atomic expressions at the profiler level. Time spent waiting for data from d-nodes will show as shallow time against a built-in (such as cts:search) or an XPath expression.

Another thing to keep in mind is that MarkLogic makes heavy use of lazy evaluation. This means that expressions may not be evaluated in the same order as shown in the code. For example, assigning the value of an expensive function call to a variable may show as taking almost no time, but at a later point, when that variable is referenced, that is where the deep time will be assigned. This is because the expensive function may not actually be invoked until the value of the variable is finally needed to satisfy evaluation of another expression.

Later versions of MarkLogic also make greater use of concurrency, which can shift things around as well. This is because, since XQuery is a functional language, multiple sub-expressions could be evaluated at the same time. This could potentially yield a deep time that is less than the sum of the deep times of the sub-expressions.

In the profiler output, it's typically the expressions with the larger shallow times that you want to look at. If an expression has a very large deep time, but almost no shallow time, then it's not actually doing very much work. All the work is being done in the sub-expressions (this is true of many FLWOR expressions, for example). Go look at those sub-expressions to see where the time is being spent. This is why profiler output is typically ordered by shallow time descending.

1. **What is diff. between cts:remainder () and xdmp:estimate() in MarkLogic?**

**Ans:**

**cts:remainder () -->** Returns an estimated search result size for a node, or of the context node if no node is provided.

**Syntax (cts:remainder ()):**

|  |  |
| --- | --- |
| **cts:remainder**( | |
|  | $node as node() |
| )  as   xs:integer | |

**Example:**

let $x := cts:search(collection(), "dog")

return

(cts:remainder($x[1]), $x)

=> ***Returns the estimated number of items in the search***

***for "dog" followed by the results of the search.***

**xdmp:estimate () -->** Returns the number of fragments selected by an expression. This can be used as a fast estimate of the number of items in a sequence.

**Syntax (xdmp:estimate()):**

|  |  |
| --- | --- |
| **xdmp:estimate**( | |
|  | $expression as item()\*, |
|  | [$maximum as xs:double?] |
| )  as   xs:integer  **$expression** : The expression to estimate. This must be a partially searchable XPath expression or a cts:search() expression.  **$maximum** (optional): The maximum value to return. Stop selecting fragments if this number is reached. | |

**Example:**

xdmp:estimate(cts:search(/bookstore/book, cts:element-attribute-value-query(xs:QName("book"),xs:QName("category"),"WEB")))

==1

(There are 2 books.xml in db, Because MarkLogic creates default fragment at “bookstore” root node where the search result contains 2 results from the same fragments)

1. **What is Triggers in MarkLogic?**

MarkLogic has the capability of firing triggers when certain events occur. These events include:

**Document events**: **create, delete, modify.**

**Property changes**.

**Database coming online**.

Triggers are stored as xml content in MarkLogic and invoke/spawn a module when an event occurs.

Triggers are used extensively by the MarkLogic Content Processing Framework to capture events and set state in the content processing framework.

**Syntax to create a trigger**:

|  |  |
| --- | --- |
| **trgr:create-trigger**( | |
|  | $trigger-name as xs:string, |
|  | $description as xs:string?, |
|  | $event as element(), |
|  | $module as element(trgr:module), |
|  | $enabled as xs:boolean, |
|  | $permissions as element(sec:permission)\*, |
|  | [$recursive as xs:boolean?], |
|  | [$task-priority as xs:string] |
| )  as   xs:unsignedLong | |

**Usage Notes:**

***Triggers must be created in the triggers database associated with the content database*** to which the triggers apply. To determine the triggers database programmatically, use xdmp:triggers-database in the evaluation context of the content database, or xdmp:database with the database name listed as the triggers database value in the Admin Interface under ***Databases > content\_db\_name***.

Triggers are created in the http://marklogic.com/xdmp/triggers/ directory, with the trigger ID completing the URI. ***This is the ID returned by trgr:create-trigger***.

If a trigger named $trigger-name already exists, the exception ***TRGR-TNEXISTS*** is raised. You must use **trgr:remove-trigger** to delete an existing trigger before recreating it.

Once a trigger is created, you can view it in the Admin Interface under the content database, in the Triggers Summary page (Databases > content\_db\_name > Triggers). If your trigger does not appear in Triggers Summary, trgr:create-trigger was probably evaluated in the wrong database context.

**Syntax to remove a trigger**:

trgr:remove-trigger(

$trigger-name as xs:string

) as empty-sequence()

**Usage Notes:**

This function deletes the specified trigger. You must run this function to remove a trigger before recreating it. If the named trigger does not exist, the exception ***TRGR-TRIGGERDNE*** is raised.

**Creating a trigger when a collection “books” is modified:**

* Create a trigger module (“/triggers/ on-book-update-post-commit.xqy”) that will be triggered on the event.

xquery version "1.0-ml";

xdmp:log("POST-COMMIT-TRIGGER","info")

* Create a trigger module (“/triggers/ on-book-update-pre-commit.xqy”) that will be triggered on the event.

xquery version "1.0-ml";

xdmp:log("PRE-COMMIT-TRIGGER","info")

* Run the given script in **qconsole**,this will create the trigger:

**Note:** **scope** could be xdmp:**collection-scope**($collname), xdmp:**document-scope**($uri-of-doc) & xdmp:**directory-scope**($dir-uri,$depth) and xdmp:**document-content**($type) action could be "***create", "modify", or "delete***".

**Post commit trigger:**

*import module namespace trgr="http://marklogic.com/xdmp/triggers" at "/MarkLogic/triggers.xqy";*

*trgr:create-trigger("book-collection-update-postcommit-trigger","Triggered book-collection-update-trigger-post-commit",*

*trgr:trigger-data-event(*

*trgr:collection-scope("books"),*

*trgr:document-content("modify"),*

*trgr:post-commit()),*

*trgr:trigger-module(xdmp:database("alfresco-modules"), "/triggers/", "on-book-update-post-commit.xqy"),*

*fn:true(), xdmp:default-permissions() )*

**Pre commit trigger:**

*import module namespace trgr="http://marklogic.com/xdmp/triggers" at "/MarkLogic/triggers.xqy";*

*trgr:create-trigger("book-collection-update-precommit-trigger","Triggered book-collection-update-trigger-pre-commit",*

*trgr:trigger-data-event(*

*trgr:collection-scope("books"),*

*trgr:document-content("modify"),*

*trgr:pre-commit()),*

*trgr:trigger-module(xdmp:database("alfresco-modules"), "/triggers/", "* *on-book-update-pre-commit.xqy"),*

*fn:true(), xdmp:default-permissions() )*

**Note:**

If using modules on file system the use following query to specify the modules database.

***trgr:trigger-module(xdmp:modules-database(), "d:\modules\triggers\", "on-book-update-post-commit.xqy")***

***It will give following result:***

***Here ‘0’ means file system.***

*<trgr:module xmlns:trgr="http://marklogic.com/xdmp/triggers">*

***<trgr:database>0</trgr:database>***

*<trgr:root>* ***d:\modules\triggers\****</trgr:root>*

*<trgr:path>on-book-update-post-commit.xqy</trgr:path>*

*</trgr:module>*

* Triggers got created. Here are the details of the triggers.

**Trigger Specification**:

For each trigger the following needs to be specified:

- **The type** of event that causes the trigger to fire - **create, modify, delete**

- **The scope** of documents to listen for events. The scope can be one of ***document, collections or directory.***

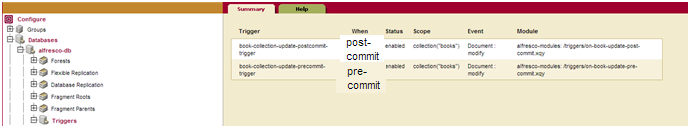
- **Type of trigger**: pre-commit or post-commit.

- **The database** on which trigger is invoked/spawned

- **XQuery module** to invoke/spawn when the event occurs.

Once a trigger has been created it cannot be modified, the trigger needs to be deleted first before recreating it.

Triggers for each database can be viewed from the admin interface at **Configure => Databases => db name => Triggers**.

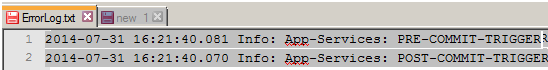


* Now run an update query on the book collection.

***xdmp:node-replace****((cts:search(/bookstore/book, cts:element-attribute-value-query(xs:QName("book"),xs:QName("category"),"WEB"))/title)[1],*

*<title lang="en">XQuery Quick Start Guide</title>)*

* Now both the triggers will be fired. See the log which we added in the scripts above.



**Pre-Commit vs. Pre-Commit Triggers**

**Pre-Commit Triggers:** Pre-Commit triggers are invoked as part of the same transaction that fired the event. If a module invoked by a trigger throws an exception then the entire transaction is rolled back, this includes the triggering event.  
To illustrate this, if a trigger is fired every time a document is created and there is an exception in the invoked module, then the document will not be created in MarkLogic.

**Post-Commit triggers:** Post commit triggers are spawned in a separate transaction to the one that fired the event. The modules to be spawned are queued on the MarkLogic task server and evaluated in the order in which they are received. There is one task server per group and can service multiple threads. To view the task server properties, use the admin interface to point to **Groups > group\_name > Task Server**.

***Execution is asynchronous*** and there is no guarantee that a module will complete. If a spawned module fails, it will not rollback the triggering transaction. So if there is a trigger on the creation of a set of documents, the documents will still be created even though the spawned module has thrown an exception.

1. **Map In MarkLogic:**

The map built-in functions are used to create maps. Maps store name-value pairs in an in-memory data structure.

You can also persist a map to disk by storing it in a document.

Some programming languages implement maps using hash tables, but these map functions make it convenient for you to create and update your own maps.

Maps are represented using the map:map XQuery primitive type.

When you serialize an object of **map:map** type, it serializes to an XML node in the http://marklogic.com/xdmp/map namespace.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | [**map:clear**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:clear) | Clear a map. | | [**map:contains**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:contains) | Returns true if the key exists in the map. (Added from ML6) | | [**map:count**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:count) | Returns the number of entries used in the map. | | [**map:delete**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:delete) | Delete a value from a map. | | [**map:get**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:get) | Get a value from a map. | | [**map:keys**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:keys) | Get the keys used in the map. | | [**map:map**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:map) | Creates a map. | | [**map:put**](file:///D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\map.html#map:put) | Put a value into a map at the given key. | |
|  |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  | | --- | |  | | |  |  | | --- | --- | | **map:clear**( | | |  | $map as map:map | | )  as   empty-sequence() | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif**Summary:**  Clear a map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:clear($table)  => () | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | |  |  | | --- | --- | | **map:contains**( | | |  | $map as map:map, | |  | $key as xs:string | | )  as   xs:boolean | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Returns true if the key exists in the map. Added from ML6 | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. |  |  | | --- | | $key : A key. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:contains($table, "some-key")  => fn:true() | |  |  |  | | --- | |  | | |  |  | | --- | --- | | **map:count**( | | |  | $map as map:map | | )  as   xs:unsignedInt | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Returns the number of entries used in the map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:count($table)  => 15 | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | |  |  | | --- | --- | | **map:delete**( | | |  | $map as map:map, | |  | $key as xs:string | | )  as   empty-sequence() | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Delete a value from a map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. |  |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $key : A key. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:delete($table, "some-key")  => () | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | |  |  | | --- | --- | | **map:get**( | | |  | $map as map:map, | |  | $key as xs:string | | )  as   item()\* | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Get a value from a map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. |  |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $key : A key. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:get($table, "some-key")  => <info>45683</info> | |  |  |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | |  |  | | --- | --- | | **map:keys**( | | |  | $map as map:map | | )  as   xs:string\* | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Get the keys used in the map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:keys($table)  => ("some-key", "another key", "and another one") | |  |  |  | | --- | |  | | |  |  | | --- | --- | | **map:map**( | | |  | [$map as element(map:map)] | | )  as   map:map | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Creates a map. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map (optional): A serialized map element. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  map:map()  => map:map(<map xmlns="http://marklogic.com/xdmp/map"/>) | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  xquery version "1.0-ml";  let $map :=  <map:map xmlns:map="http://marklogic.com/xdmp/map">  <map:entry key="2">  <map:value xsi:type="xs:string"  xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  >world</map:value>  </map:entry>  <map:entry key="1">  <map:value xsi:type="xs:string"  xmlns:xs="http://www.w3.org/2001/XMLSchema"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  >hello</map:value>  </map:entry>  </map:map>  return  map:get(map:map($map), "2")  =>  world | |  |  |  | | --- | |  | | |  |  | | --- | --- | | **map:put**( | | |  | $map as map:map, | |  | $key as xs:string, | |  | $value as item()\* | | )  as   empty-sequence() | | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Summary:**  Put a value into a map at the given key. | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Parameters:**   |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $map : A map. |  |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $key : A key. If the key is not unique, it will overwrite the existing key. |  |  | | --- | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $value : A value. If the value is the empty sequence, it will remove the key from the map. | | | D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif **Example:**  let $map := map:map()  let $put := map:put($map, "some-key",  <info>45683</info>)  return  <result>{$map}</result>  =>  <result>  <map:map xmlns:map="http://marklogic.com/xdmp/map"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:xs="http://www.w3.org/2001/XMLSchema">  <map:entry key="some-key">  <map:value>  <info>45683</info>  </map:value>  </map:entry>  </map:map>  </result> | | |

1. **xdmp:document-insert() vs xdmp:document-load():**

**xdmp:document-insert() -->**

Inserts a new document into the database if a document with the specified URI does not already exist. If a document already exists at the specified URI, the function replaces the content of the existing document with the specified content (the $root parameter) as an update operation.

In addition to replacing the content, xdmp:document-insert replaces any permissions, collections, and quality with the ones specified (or with the default values for these parameters, if not explicitly specified).

Also***, if a properties document exists at the same URI, that properties document (including any content it contains) is preserved.***

***The specified node ($root) is an in-memory node that will be saved in the database.***

**Syntax:**

|  |  |
| --- | --- |
| **xdmp:document-insert**( | |
|  | $uri as xs:string, |
|  | **$root as node(),** |
|  | [$permissions as element(sec:permission)\*], |
|  | [$collections as xs:string\*], |
|  | **[$quality as xs:int?**], |
|  | [$forest-ids as xs:unsignedLong\*] |
| )  as   empty-sequence()  **$root:** The root node. The root node can be one of XML format, binary (BLOB) format, or text (CLOB) format. And $root will be in memory node.  **$quality (optional):** A positive value increases the relevance score of the document in text search functions. The converse is true for a negative value. The default value is 0. | |

**Example:**

(: With default permissions :)

*xdmp:document-insert("/example.xml",<a>Hello</a>,*

*xdmp:default-permissions(),"example",10)*

(: With custom permissions :)

*xdmp:document-insert("/example.xml",<a>Hello</a>,*

*(xdmp:permission("editor", "read"),xdmp:permission("editor", "update")),*

*"example",10)*

**xdmp:document-load() --> (xdmp:document-get()+xdmp:document-insert())**

Inserts a new document with the specified URI. It will fetch the specified document from the given uri (filesystem uri) or http url and then loads it. If a document already exists at the URI, the function replaces the content in the existing document as an update operation.

***If the URI is the filesystem uri then internally it will use xdmp:document-get($URI) to fetch the document. xdmp:document-get(..) will return the node.***

***If the URI is the http uri then internally it will use xdmp:http-get($URI) to fetch the document.***

**Syntax:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **xdmp:document-load**( | | |  | $location as xs:string, | |  | [$options as node()?] | | )  as   empty-sequence() | | |
| D:\Ebooks\MarkLogic\MarkLogic_6_pubs\pubs\apidocs\images\varspace.gif  $location -- uri  $option -- <options></options> |

**Example:**

*xdmp:document-load("c:\myFile.xml",*

***<options xmlns="xdmp:document-load">***

*<uri>/documents/myFile.xml</uri>*

*<permissions>{xdmp:default-permissions()}</permissions>*

*<collections>*

*<collection>myCollection1</collection>*

*<collection>myCollection2</collection>*

*</collections>*

*<format>xml</format>*

*<repair>full</repair>*

*<forests>*

*<forest>{xdmp:forest("myForest")}</forest>*

*</forests>*

*</options> )*

***=> Loads the document with a URI "/documents/myFile.xml"***

***performing tag repair during the load, adding the***

***document to the "myCollection1" and "myCollection2"***

***collections, and loading the document into the forest***

***named "myForest".***

*xdmp:document-load("http://myCompany.com/file.xml",*

*<****options xmlns="xdmp:document-load" xmlns:http="xdmp:http****">*

*<uri>/documents/myFile.xml</uri>*

*<repair>none</repair>*

*<permissions>{xdmp:default-permissions()}</permissions>*

*<format>xml</format>*

***<http:authentication>***

***<http:username>user</http:username>***

***<http:password>pass</http:password>***

***</http:authentication>***

*</options>)*

***=> Loads the document with a URI "/documents/myFile.xml"***

***from the server http://myCompany.com, sending the***

***credentials user/pass. Tag repair is not performed***

***during the load, and the document is loaded as xml.***

**The xdmp:document-load options include:**

**<uri>**

The URI of the document to be loaded. If omitted, then the location is used for the URI.

**<permissions>**

Security permission elements corresponding to the permissions for the document. ***If not supplied, the current user's default permissions are applied.*** The default value used for $permissions can be obtained by calling xdmp:default-permissions(). A document that is created by a non-admin user (that is, by any user who does not have the admin role) must have at least one update permission, otherwise the creation will throw an ***XDMP-MUSTHAVEUPDATE exception***.

**<collections>**

The collection URIs for collections to which this document belongs. If not supplied, the document is added to the current user's default collections (the collections returned from xdmp:default-collections()). For each collection that is protected, the user must have permissions to update that collection or have the any-collection privilege. For each unprotected collection, the user must have the unprotected-collections privilege. The <collections> element consists of one or more <collection> child elements. For example:

<collections>

<collection>myCollection1</collection>

<collection>myCollection2</collection>

</collections>

**<quality>**

The quality of this document. A positive value increases the relevance score of the document in text search functions. The converse is true for a negative value. The default value is 0.

**<default-namespace>**

The namespace to use if there is no namespace at the root node of the document. The default value is "".

**<repair> {full and none}**

A value of **full** specifies that malformed XML content be repaired. A value of **none** specifies that malformed XML content is rejected. If no repair option is explicitly specified, the default is implicitly specified by the XQuery version of the caller. ***In XQuery 1.0 and 1.0-ml the default is none***. In ***XQuery 0.9-ml the default is full***. This option ***has no effect on binary or text documents***.

**<format> {text, binary and xml}**

A value of **text** specifies to get the document as a text document, regardless of the URI specified. A value of **binary** specifies to get the document as a binary document, regardless of the URI specified. A value of **xml** specifies to get the document as an XML document, regardless of the URI specified.

**<default-language>**

The language to specify in an ***xml:lang*** attribute on the root element node if the root element node does not already have an ***xml:lang*** attribute. If default-language is not specified, then nothing is added to the root element node.

**<forests>**

Specifies the ID of the forest in which this document is inserted. Each forest ID is in a <forest> child element and is of type xs:unsignedLong. If the document already exists in the database, it will remain in its existing forest. If no such forest exists or if no such forest is attached to the context database, an error is raised. If multiple forests are specified, the document is inserted into one of the specifed forests. If the document already exists and the forest in which it is stored is set to delete-only, then you must specify the forest IDs to include one or more forests that allow updates, otherwise an exception is thrown.

If you have local disk failover enabled, specify the ID of the master forest. In the event of a failover, MarkLogic server will automatically redirect documents to the replica forest. Specify the ID of the replica forest will result in a "forest not in database" error.

**<encoding> {Marklogic transform all encoding to UTF-8 in itself}**

Specifies the encoding to use when reading the document into MarkLogic Server. Supported values include UTF-8, ISO-8859-1, as well as many other popular encodings. See the Search Developer's Guide for a list of character set encodings by language. ***All encodings will be translated into UTF-8 from the specified encoding.*** The string specifed for the encoding option will be matched to an encoding name according to the Unicode Charset Alias Matching rules (http://www.unicode.org/reports/tr22/#Charset\_Alias\_Matching). An automatic encoding detector will be used if the value auto is specified. If no encoding can be detected, the encoding defaults to UTF-8. If no encoding option is specified and you are using a path with an http:// scheme, the encoding defaults to the encoding specified in the http header (if an encoding header is specified), otherwise it defaults to UTF-8.

1. **What needs to be enabled to profile an Xquery program?**

**Ans:**

To profile an XQuery program, set the **profile allow option to true** in the **App Server** or **task server** in which the request is serviced.

If you request a **profile report** for an XQuery program whose App Server does not have profiling enabled (and **if prof:enable** is not called before generating the profile report), then the profile APIs do not profile the request, and the APIs that generate the report return the empty sequence.

If you request prof:eval, then a <prof:report> node will be prepended to any output produced by evaluating the string. If profiling is not allowed for the App Server, this function will throw a

**PROF-PROFILEALLOW** exception.

1. **What is the syntax for Importing and declaring a module?**

**Declaring a module (test.xqy)**:

module namespace test = “www.example.com/test”;

**Importing a module (test-main.xqy)**:

import module namespace test = “www.example.com/test” at “/test.xqy”;

1. **What is the syntax for declaring default namespace?**

xquery version "1.0-ml";

declare namespace test = “www.example.com/test”;

1. **What is the syntax for declaring default schema or element namespace?**

xquery version "1.0-ml";

declare default element namespace "my.namespace.hello";

**Then**

***<some-element xmlns="my.namespace.hello">element content with default namespace goes here</some-element>***

1. **cts:search gives filtered result or unfiltered?**

By default search is filtered, you can add “unfiltered” as option to get the unfiltered search.

1. **cts:element-range-query and need of indexes ?**

**Ans:**

Returns a cts:query matching elements by name with a range-index entry equal to a given value.

Searches with the cts:element-range-query constructor require an element range index on the specified QName(s);

If there is no range index configured, then an exception is thrown.

**Syntax:**

**cts:element-range-query**($element-name as xs:QName\*,

$operator as xs:string,

$value as xs:anyAtomicType\*,

[$options as xs:string\*],

[$weight as xs:double?])

as cts:element-range-query

**Example**:

cts:search(/pce:IC, cts:element-range-query(xs:QName("dc:identifier"),"=",13669626843246594231))

Note: Here an element range index of type xs:unsignedLong is created for identifier element.

**$element-name:** One or more element QNames to match. When multiple QNames are specified, the query matches if any QName matches.

**$operator:** A comparison operator.

**Operators include**:

"<"

Match range index values less than $value.

"<="

Match range index values less than or equal to $value.

">"

Match range index values greater than $value.

">="

Match range index values greater than or equal to $value.

"="

Match range index values equal to $value.

"!="

Match range index values not equal to $value.

**$value**: One or more element values to match. When multiple values are specified, the query matches if any value matches.

**$options (optional):** Options to this query. The default is ().

**Options include:**

**"collation=URI"**

Use the range index with the collation specified by URI. If not specified, then the default collation from the query is used. ***If a range index with the specified collation does not exist, an error is thrown.***

**"cached"**

Cache the results of this query in the ***list cache***.

**"uncached"**

Do not cache the results of this query in the list cache.

**"min-occurs=number"**

Specifies the minimum number of occurrences required. If fewer that this number of words occur, the fragment does not match. ***The default is 1.***

"**max-occurs=number"**

Specifies the maximum number of occurrences required. If more than this number of words occur, the fragment does not match. ***The default is*** ***unbounded***.

**"synonym"**

Specifies that all of the terms in the $text parameter are considered synonyms for scoring purposes. The result is that occurrences of more than one of the synonyms are scored as if there are more occurrence of the same term (as opposed to having a separate term that contributes to score).

**Note**: ***If neither "cached" nor "uncached" is present, it specifies "cached",*** and

***Negative "min-occurs" or "max-occurs" values will be treated as 0*** and non-integral values will be rounded down. An ***error will be raised if the "min-occurs" value is greater than the "max-occurs" value.***

**$weight (optional):** A weight for this query. The default is 1.0. In the current release, this option is ignored; range queries do not contribute to the score.

***An Element Range Index accelerates queries for comparisons within a specified type.***

***Each Element Range Index keeps track of the values appearing in all elements with a given name, type, and collation.***

Element Range Indexes also allow you to use the ***cts:element-values*** family of lexicon APIs and to use the ***cts:element-range-query*** constructor in searches.

1. **cts:word-query?**

**Ans:**

Returns a query matching text content containing a given phrase.

**Syntax:**

**cts:word-query**(

$text as xs:string\*,

[$options as xs:string\*],

[$weight as xs:double?]

) as cts:word-query

**Parameters:**

**$text :** Some words or phrases to match. When multiple strings are specified, the query matches if any string matches.

**$options (optional):** Options to this query. The default is ().

**Options include:**

"case-sensitive"

A case-sensitive query.

"case-insensitive"

A case-insensitive query.

"diacritic-sensitive"

A diacritic-sensitive query.

"diacritic-insensitive"

A diacritic-insensitive query.

"punctuation-sensitive"

A punctuation-sensitive query.

"punctuation-insensitive"

A punctuation-insensitive query.

"whitespace-sensitive"

A whitespace-sensitive query.

"whitespace-insensitive"

A whitespace-insensitive query.

"stemmed"

A stemmed query.

"unstemmed"

An unstemmed query.

"wildcarded"

A wildcarded query.

"unwildcarded"

An unwildcarded query.

***"exact"***

***An exact match query. Shorthand for "case-sensitive", "diacritic-sensitive", "punctuation-sensitive", "whitespace-sensitive", "unstemmed", and "unwildcarded".***

"lang=iso639code"

Specifies the language of the query. The iso639code code portion is case-insensitive, and uses the languages specified by ISO 639. The default is specified in the database configuration.

"distance-weight=number"

A weight applied based on the minimum distance between matches of this query. Higher weights add to the importance of proximity (as opposed to term matches) when the relevance order is calculated. The default value is 0.0 (no impact of proximity). The weight should be less than or equal to the absolute value of 16 (between -16 and 16); weights greater than 16 will have the same effect as a weight of 16. This parameter has no effect if the word positions index is not enabled. This parameter has no effect on searches that use score-simple, score-random, or score-zero (because those scoring algorithms do not consider term frequency, proximity is irrelevant).

"min-occurs=number"

Specifies the minimum number of occurrences required. If fewer that this number of words occur, the fragment does not match. The default is 1.

"max-occurs=number"

Specifies the maximum number of occurrences required. If more than this number of words occur, the fragment does not match. The default is unbounded.

"synonym"

Specifies that all of the terms in the $text parameter are considered synonyms for scoring purposes. The result is that occurrences of more than one of the synonyms are scored as if there are more occurrence of the same term (as opposed to having a separate term that contributes to score).

**$weight (optional):** A weight for this query. Higher weights move search results up in the relevance order. The default is 1.0. The weight should be less than or equal to the absolute value of 16 (between -16 and 16); weights greater than 16 will have the same effect as a weight of 16. Weights less than the absolute value of 0.0625 (between -0.0625 and 0.0625) are rounded to 0, which means that they do not contribute to the score.

**Usage Notes:**

* If neither "case-sensitive" nor "case-insensitive" is present, $text is used to determine case sensitivity. If $text contains no uppercase, it specifies "case-insensitive". If $text contains uppercase, it specifies "case-sensitive".
* If neither "diacritic-sensitive" nor "diacritic-insensitive" is present, $text is used to determine diacritic sensitivity. If $text contains no diacritics, it specifies "diacritic-insensitive". If $text contains diacritics, it specifies "diacritic-sensitive".
* If neither "punctuation-sensitive" nor "punctuation-insensitive" is present, $text is used to determine punctuation sensitivity. If $text contains no punctuation, it specifies "punctuation-insensitive". If $text contains punctuation, it specifies "punctuation-sensitive".
* If neither "whitespace-sensitive" nor "whitespace-insensitive" is present, the query is "whitespace-insensitive".
* If neither "wildcarded" nor "unwildcarded" is present, the database configuration and $text determine wildcarding. If the database has any wildcard indexes enabled ("three character searches", "two character searches", "one character searches", or "trailing wildcard searches") and if $text contains either of the wildcard characters '?' or '\*', it specifies "wildcarded". Otherwise it specifies "unwildcarded".
* If neither "stemmed" nor "unstemmed" is present, then the database configuration determines if a query is run as "stemmed" (stemmed searches enabled) or "unstemmed" (word searches enabled and stemmed searches disabled). If the query is a wildcard query and is also a phrase query (contains two or more terms), then any wildcard terms in the query will be "unstemmed".
* Negative "min-occurs" or "max-occurs" values will be treated as 0 and non-integral values will be rounded down. An error will be raised if the "min-occurs" value is greater than the "max-occurs" value.
* Relevance adjustment for the "distance-weight" option depends on the closest proximity of any two matches of the query. For example,

***cts:word-query(("dog","cat"),("distance-weight=10"))***

will adjust relevance based on the distance between the closest pair of matches of either "dog" or "cat" (the pair may consist only of matches of "dog", only of matches of "cat", or a match of "dog" and a match of "cat").

**Example:**

cts:search(/bookstore/book, cts:word-query(("Gucategorye","Ita\*"),("case-sensitive")))

1. **cts:element-walk? (Added after ML6)**

Same as cts:walk but returns a copy of the node, replacing any elements found with the specified expression.

**Syntax**:

***cts:element-walk(***

[***$node***](https://docs.marklogic.com/6.0/cts:element-walk?q=cts:element-walk#node) ***as node(),***

[***$element***](https://docs.marklogic.com/6.0/cts:element-walk?q=cts:element-walk#element) ***as xs:QName\*,***

[***$expr***](https://docs.marklogic.com/6.0/cts:element-walk?q=cts:element-walk#expr) ***as item()\****

***) as node()***

**Example:**

let $bookstore := <bookstore>

<book category="bk101">

<author>Gambardella, Matthew</author>

<title>Giada</title>

<genre>Computer</genre>

<price>44.95</price>

<publish\_date>2000-10-01</publish\_date>

<description>An in-depth look at creating applications

with XML.</description>

</book>

<book category="bk102">

<author>Anderson</author>

<title>Java</title>

<genre>Computer</genre>

<price>44.95</price>

<publish\_date>2000-10-01</publish\_date>

<description>An in-depth look at java</description>

</book>

</bookstore>

return

cts:element-walk($bookstore, xs:QName("title"), <newTitle>{$cts:node/text()}</newTitle>)

**Output>>**

<bookstore>

<book category="bk101">

<author>Gambardella, Matthew</author>

***<newTitle>Giada</newTitle>***

<genre>Computer</genre>

<price>44.95</price>

<publish\_date>2000-10-01</publish\_date>

<description>An in-depth look at creating applications

with XML.</description>

</book>

<book category="bk102">

<author>Anderson</author>

<newTitle>Java</newTitle>

***<genre>Computer</genre>***

<price>44.95</price>

<publish\_date>2000-10-01</publish\_date>

<description>An in-depth look at java</description>

</book>

</bookstore>

1. **Who to separate a transaction?**

**Ans:**

A transaction can be separated using “;” keyword.

1. **cts:register , cts:registered-query & cts:deregister?**

**Ans:**

**cts:register >>**

Register a query for later use.

**Syntax:**

**cts:register**($query as cts:query) as xs:unsignedLong

**Example:**

cts:register(cts:word-query("Gucategorye"))

=> 16244301519510021360

**cts:registered-query >>**

Returns a query matching fragments specified by previously registered queries (see cts:register). If a registered query with the specified ID(s) is not found, then a cts:search operation with an invalid cts:registered-query throws an XDMP-UNREGISTERED exception.

Registered queries require the "unfiltered" option.

**Options include:**

"unfiltered"

An unfiltered query. Unfiltered registered queries select fragments from the indexes that are candidates to satisfy the cts:query. Depending on the original cts:query, the structure of the documents in the database, and the configuration of the database, unfiltered registered queries may result in false-positive results or in incorrect matches when there are multiple candidate matches within the same fragment. To avoid these problems, you should only use unfiltered queries on top-level XPath expressions (for example, document nodes, collections, directories) or on fragment roots. Using unfiltered queries on complex XPath expressions or on XPath expressions that traverse below a fragment root can result in unexpected results. This option is required in the current release.

**Usage Notes:**

If the options parameter does not contain "unfiltered", then an error (***XDMP-REGFLT***) is returned, as the "unfiltered" option is required.

Registered queries are persisted as a soft state only; they can become unregistered through an explicit direction (using cts:deregister), as a result of the cache growing too large, or because of a server restart. Consequently, either your XQuery code or your middleware layer should handle the case when an ***XDMP-UNREGISTERED*** exception occurs (for example, you can wrap your cts:registered-query code in a try/catch block or your Java or .NET code can catch and handle the exception).

**Example:**

(: wrap the registered query in a try/catch :)

try {

cts:search(/bookstore/book,cts:registered-query(16244301519510021360,"unfiltered")))

} catch ($e) {

if ($e/err:code = "XDMP-UNREGISTERED")

then ("Retry this query with the following registered query ID: ",

cts:register(cts:word-query("Gucategorye")))

else $e

}

**cts:deregister>>**

Deregister a registered query, explicitly releasing the associated resources.

**Syntax:**

cts:deregister(

$id as xs:unsignedLong

) as empty-sequence()

**Example:**

cts:deregister(xs:unsignedLong("12345678901234567"))

=> ()

1. **xdmp:function & xdmp:apply?**

**xdmp:function>>**

Returns a function value as an xdmp:function type. You can return an xdmp:function from an expression or a function. You can execute the function referred to by an xdmp:function by passing the xdmp:function value to xdmp:apply.

**Syntax:**

|  |  |
| --- | --- |
| **xdmp:function**( | |
|  | $function as xs:QName, |
|  | [$module-path as xs:string?] |
| )  as   xdmp:function | |

**Example:**

xquery version "1.0-ml";

declare namespace admin="http://marklogic.com/xdmp/admin";

xdmp:function(xs:QName("admin:get-configuration"), "/MarkLogic/admin.xqy")

**Example:**

xquery version "1.0-ml";

let $function := xdmp:function(xs:QName("fn:concat"))

return

xdmp:apply($function, "hello", " world")

=> hello world

**xdmp:apply>>**

Applies an xdmp:function with the given parameters.

**Syntax:**

xdmp:apply(

$function as xdmp:function,

[$params-1-to-N as item()\*]

) as item()\*

**Parameters:**

**$function :** The xdmp:function value to be applied.

**$params-1-to-N (optional):** The parameters to pass into the specified function value. Specify one parameter for each parameter that the specified function takes, with the first parameter corresponding to the first parameter in the specified function's signature, the second parameter corresponding to the second, and so on. Omit this parameter if the specified function takes no parameters.

**Example:**

let $function := xdmp:function(xs:QName("fn:concat"))

return

xdmp:apply($function, "hello", " world")

=> hello world

**Example:**

let $function := xdmp:function(xs:QName("fn:current-date"))

return

xdmp:apply($function)

=> 2009-02-14-08:00 (or whatever is the current date)

**Note:** If you pass a $function which is not defined in xdmp:apply then ***XDMP-UNDFUN*** error will be trown.

1. **xdmp:server() , xdmp:server-name(), xdmp:servers()?**

**Ans:**

**xdmp:server()>>>**

Returns the ID(s) of the App Server specified in the parameter. Returns the ID of the current App Server if no parameter is specified.

**Syntax:**

|  |  |
| --- | --- |
| **xdmp:server**( | |
|  | [$name as xs:string], |
|  | [$group as xs:unsignedLong] |
| )  as   xs:unsignedLong+ | |

**Parameters:**

**$name (optional):** The name of the App Server. The default value is the name of the current App Server.

**$group (optional):** The group of the App Server. If not specified, the ids of the named App Server in each group is returned.

**Example:**

xdmp:server("test-http")

=> 15495681647217162987

***xdmp:server() => 11440694566445514982***

***If no param provided it will return ID of the current app server, If this is executed in qconsole then the Id will be the id of app services http server.***

**xdmp:server-name()>>>**

Return the name of the App Server with the given ID.

**Syntax:**

|  |  |
| --- | --- |
| **xdmp:server-name**( | |
|  | $id as xs:unsignedLong |
| )  as   xs:string | |

**$id :** app server id

**Example:**

xdmp:server-name(11440694566445514982)

=> App-Services

**xdmp:servers()>>>**

Returns a sequence of the IDs of all the App Servers in the system.

**Syntax:**

**xdmp:servers( ) as xs:unsignedLong\***

**Example:**

xdmp:servers()

* 11440694566445514982

10988695772589824455

14481631097670642241

12083521916691424327

10325916934242540239

**Example:**

for $each-server in xdmp:servers()

return xdmp:server-name($each-server)

=>

App-Services

Admin

Manage

HealthCheck

TaskServer

1. **xdmp:quote() and xdmp:unquote() ?**

**Ans:**

**xdmp:quote() >>**

Returns the unevaluated serialized representation of the input parameter as a string.

**Syntax:**

xdmp:quote(

$arg as item()\*,

[$options as node()?]

) as xs:string

**The xdmp:quote options include:**

**<output-encoding>**

Specifies the encoding to use for this quote operation. This is only used to escape characters that cannot be represented.

**<output-sgml-character-entities>**

Specifies if character entities should be output upon serialization of the XML. Valid values are normal, none, math, and pub. By default (that is, if this option is not specified), no SGML entities are

serialized on output, unless the App Server is configured to output SGML character entities.

**<method>**

Valid values are xml, html, xhtml, and text. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<cdata-section-elements>**

A list of QNames to output as CDATA sections. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<encoding>**

The encoding. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<use-character-maps>**

Valid values are xdmp:sgml-entities-normal, xdmp:sgml-entities-math, and xdmp:sgml-entities-pub. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<media-type>**

A mimetype representing a media type. For example, text/plain or text/xml (or other valid mimetypes). This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<byte-order-mark>**

Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<indent>**

Specifies if typed XML (that is, XML for which there is an in-scope schema) should be pretty-printed (indented). Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery **xdmp:output** prolog statement.

**<indent-untyped>**

Specifies if untyped XML (that is, XML for which there is no in-scope schema) should be pretty-printed (indented). Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<include-content-type>**

Include the content-type declaration when serializing the node. Valid values are yes or no.

**<escape-uri-attributes>**

Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<doctype-public>**

A public identifier, which is the public identifier to use on the emitted DOCTYPE. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<doctype-system>**

A system identifier, which is the system identifier to use on the emitted DOCTYPE. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<omit-xml-declaration>**

Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<standalone>**

Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<normalization-form>**

Valid values are NFC, NFD, and NFKD. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**<default-attributes>**

Specifies whether attributes defaulted with a schema should be included in the serialization. Valid values are yes or no. This is like the corresponding part of both the XSLT xsl:output instruction and the MarkLogic XQuery xdmp:output prolog statement.

**Example:**

let $arg := <a>aaa</a>

return ($arg, xdmp:quote($arg))

=> (<a>aaa</a>, "<a>aaa</a>") – As of ML6/7

let $arg := <a> </a>

return ($arg, xdmp:quote($arg))

=> (<a/>, "<a/>") – As of ML6/7

**Example ML5:**

let $arg := <a>aaa</a>

return ($arg, xdmp:quote($arg))

=> (<a>aaa</a>&lt;a&gt;aaa&lt;/a&gt;) – As of ML5

let $arg := <a> </a>

return ($arg, xdmp:quote($arg))

=> (<a/>&lt;a/&gt;) – As of ML5

**Example as of ML7:**

***xquery version "1.0-ml";***

***declare option xdmp:output "indent-untyped=yes";***

***declare option xdmp:output "omit-xml-declaration=no";***

***xdmp:quote(/bookstore[1])***

* Returned the below xml in string format.

<?xml version="1.0" encoding="UTF-8"?>  
<bookstore>  
<book category="bk101">  
<author>Gambardella, Matthew</author>  
<title>XML Developer's Gucategorye</title>  
<genre>Computer</genre>  
<price>44.95</price>  
<publish\_date>2000-10-01</publish\_date>  
<description>An in-depth look at creating applications   
with XML.</description>  
</book>

</bookstore>

**xdmp:unquote() >>**

Parses a string as XML, returning one or more document nodes.

**Syntax:**

xdmp:unquote(

$arg as xs:string,

[$default-namespace as xs:string?],

[$options as xs:string\*]

) as document-node()+

**Options include:**

**"repair-full"**

Specifies that malformed XML content be repaired. XML content with multiple top-level elements will be parsed as multiple documents. This option has no effect on binary or text documents.

**"repair-none"**

Specifies that malformed XML content be rejected. XML content will be parsed as a single document, so a maximum of one document node will be returned. This option has no effect on binary or text documents.

**"format-text"**

Specifies to get the document as a text document, regardless of the URI specified.

**"format-binary"**

Specifies to get the document as a binary document, regardless of the URI specified.

**"format-xml"**

Specifies to get the document as an XML document, regardless of the URI specified.

**"default-language=xx"**

If the root element node specified in the first parameter does not already have an xml:lang attribute, the language to specify in an xml:lang attribute on the root element node. If default-language is not specified, then nothing is added to the root element node. Some examples are default-language=en and default-language=fr.

**Usage Notes:**

***If no format is specified in $options, it is XML.***

***If neither "repair-full" nor "repair-none" is present, the default is specified by the XQuery version of the caller. In XQuery version 1.0 and 1.0-ml the default is "repair-none". In XQuery version 0.9-ml the default is "repair-full".***

***If $arg is the empty string (xdmp:unquote(“”)) , xdmp:unquote returns an empty document node.***

**Example:**

xdmp:unquote("<foo/>")

=> <foo/>

It returns this as a document node.

**Example:**

xdmp:unquote('<foo>hello</foo>', "",

("repair-none", "default-language=en"))

=> <foo xml:lang="en">hello</foo>

It returns this as a document node and does not perform tag repair on the node.

**Example:**

xdmp:unquote('<foo>hello</foo>', "http://example.com",

("repair-none", "default-language=en"))

=> <foo xml:lang="en" xmlns=”http://example.com”>hello</foo>

1. **cts:contains () ?**

Returns true if any of a sequence of nodes matches a query.

**Syntax:**

cts:contains(

$nodes as item()\*,

$query as cts:query

) as xs:boolean?

**Example:**

let $bookstore := <bookstore>

<book category="JAVA">

<title lang="en">Java Basics</title>

<author>AbhiKMishra</author>

<year>2005</year>

<price>300.00</price>

</book>

<book category="DOTNET">

<title lang="en">DOT NET BLACK BOOK</title>

<author>J K. **Rowling**</author>

<year>2005</year>

<price>200.99</price>

</book>

</bookstore>

return

***cts:contains***($bookstore,cts:element-word-query(xs:QName("author"),"**Rowling**"))

* returns “true”

1. **Flower:**

***FLWOR is an acronym for "For, Let, Where, Order by, Return".***

* The **for** clause selects all book elements under the bookstore element  into a variable called $x.
* The **where** clause selects only book elements with a price element with a value greater than 30.
* The **order by** clause defines the sort-order. Will be sort by the title element.
* The **return** clause specifies what should be returned. Here it returns the title elements.

***for $i in fn:doc("/books.xml")/bookstore/book***

***let $price:=$i/price***

***where $price>40***

***order by $i/price descending (:ascending:)***

***return $price***

1. **Predicate:**

***XQuery uses predicates to limit the extracted data from XML documents.***

The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30:

**fn:doc("books.xml")/bookstore/book[price<30]**

1. **Meaning for \*, ? :**

MarkLogic Server supports two wildcard characters: \* and ?.

* \* matches zero or more non-space characters. (he\*== here,hello,help,helicopter etc.)
* ? matches exactly one non-space character. (he?== hen,her etc.)

For example, he\* will match any word starting with he, such as he, her, help, hello, helicopter, and so on. On the other hand, he? will only match three-letter words starting with he, such as hem, hen, and so on.

1. **Collection vs. directory:**

**-** Collection is virtual where directory is physical.

**-** One document can be mapped in many collection but will exist physically in 1 directory.

**-** URI of collection for an xml could be anything but for an xml in a directory should start with the directory name separated by ‘/’ with document name.

1. **xdmp:estimate** **vs. fn:count:**

* fn:count is Accurate, xdmp:estimate is Fast.
* fn:count can return count of on the fly created nodes, where xdmp:estimate can’t because it always looks for indexes.
* fn:count, which frequently must process its answer by inspecting the data directly (hence the heavy I/O loads), xdmp:estimate computes its answer directly from indexes.
* In instances where xdmp:estimate is not able to return a fast estimate, it will throw an error. Hence, you can depend on xdmp:estimate to be fast, just as you can depend on fn:count to be accurate.
* xdmp:estimate does not always return the same value as fn:count. The fn:count function returns the exact number of items in the sequence that is provided as a parameter.
* ***cts:remainder($node):***

*-* Returns an estimated search result size for a node, or of the context node if no node is provided. It can only be used with CTS:QUERY.

- The search result size for a node is the number of fragments remaining (including the current node) in the result sequence containing the node. This is useful to quickly estimate the size of a search result sequence, without using fn:count() or xdmp:estimate().

*let $x := cts:search(collection(), "dog")*

*return (cts:remainder($x[1]), $x)*

*=> Returns the estimated number of items in the search*

*for "dog" followed by the results of the search.*

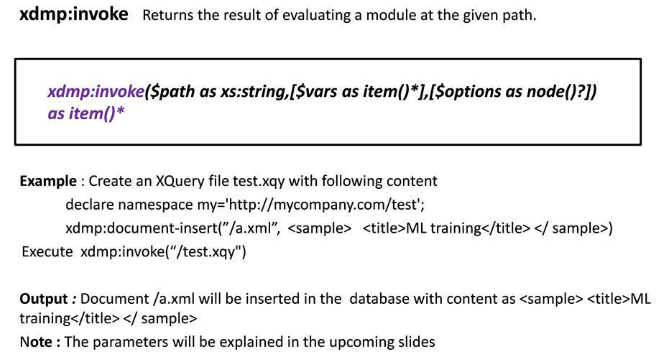
1. **Indexing**: Indexes in ML are same as the indexes in relational databases.

It is created for improved search performance.

- MarkLogic creates indexes for unique words and elements by default while content load called as “Universal indexes”.  
- For range queries (comparison queries) we can create “Element range indexes, Attribute range indexes and Field range indexes”.

- Field range indexes can be used to specify the fields to be included and excluded the elements during the search and allow to fire range query on them as well.

1. **Sequences:** It is a kind of array in XQuery.
2. **How many times a predicate executed:** once (for a simple let expression), but depends on query and loops. **For example** if u are writing a predicate on multi book elements having title element then it will check for each book with the given title and return the matched one.
3. **Difference between invoke, eval and spawn functions:**

****

**Example:**

**//Create an xquery file “authHelper.xqy”.**

declare variable $userName as xs:string \* external;

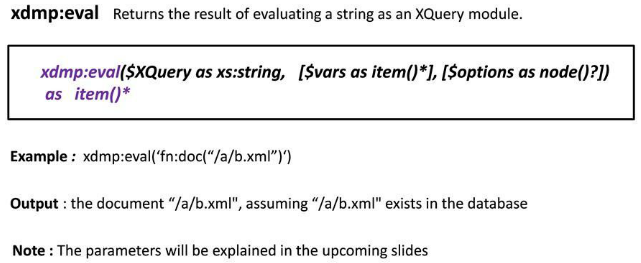
declare variable $password as xs:string \* external;

xdmp:login($userName,$password)

**--Passing external values**

xdmp:invoke("/authHelper.xqy",(xs:QName("userName"),"admin",xs:QName("password"),"passw0rd"))

**Note: We cannot invoke library modules, we can invoke main modules only. It will throw exception “*XDMP-EVALLIBMOD*” when you try to invoke library module.**

****

**Example:**

**let $queryAsString :=** "declare variable $a as xs:integer external;

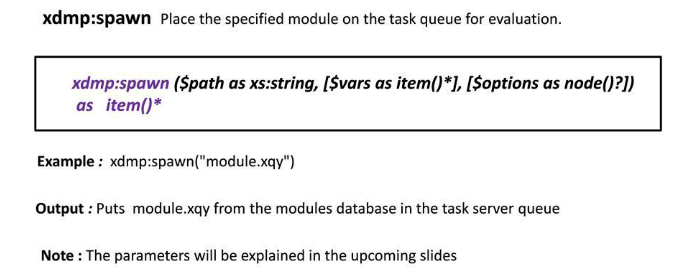
declare variable $b as xs:integer external;

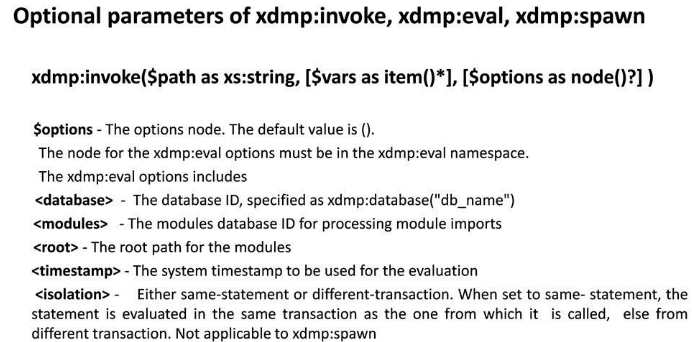
($a+$b)

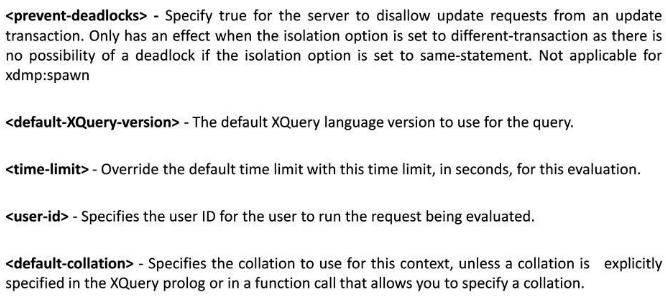
"

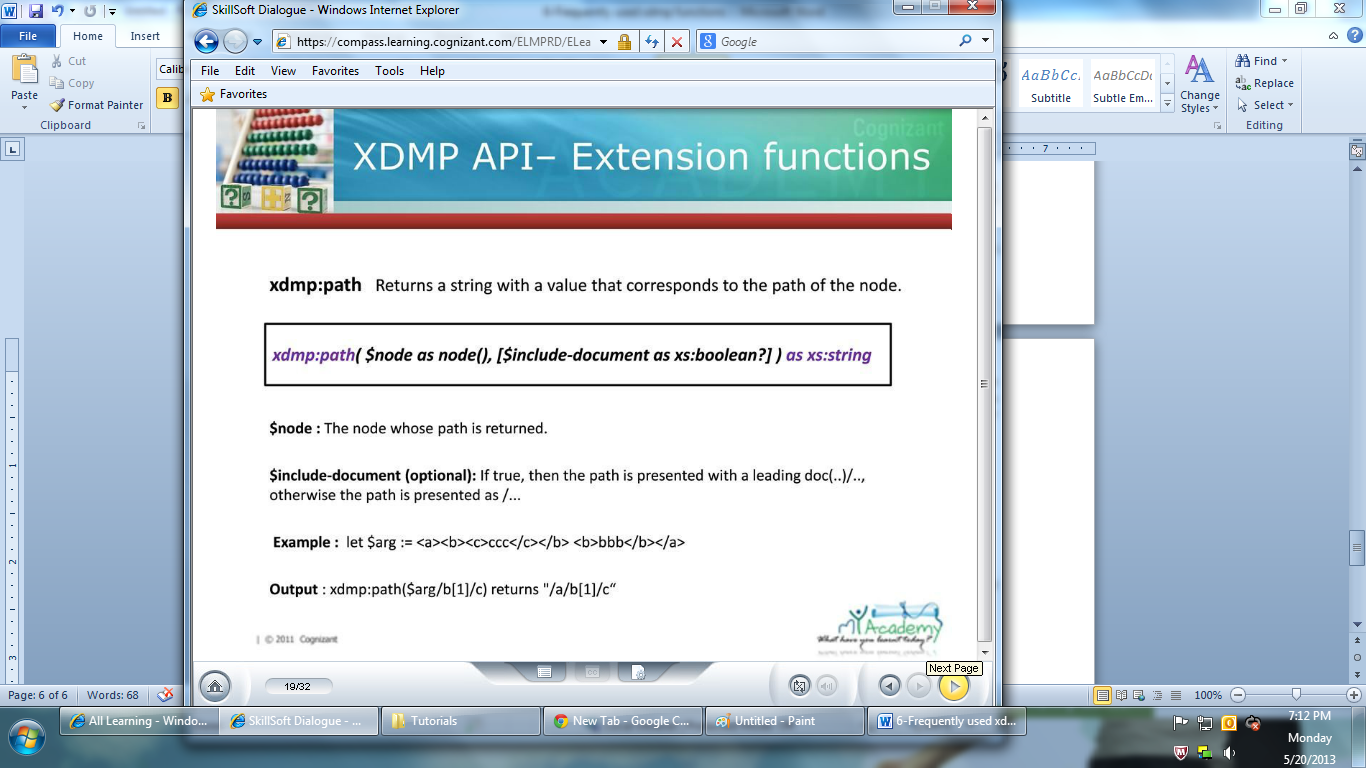
**return** xdmp:eval($queryAsString,(xs:QName("a"),10,xs:QName("b"),50))

**Output>> 60**

****

****

****



**Example:**

xdmp:path(fn:doc("/books.xml")/books/book)

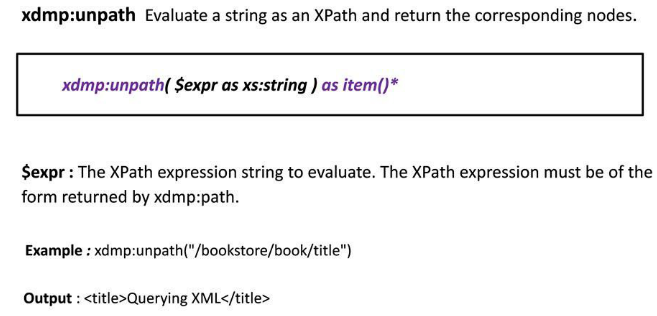
**Output>**

/books/book[1]

/books/book[2]

/books/book[3]

/books/book[4]

****

**Example:**

Xdmp:unpath(/books/book[1] )

Xdmp:unpath(/books/book[2] )

Xdmp:unpath(/books/book[3] )

Xdmp:unpath(/books/book[4])

Xdmp:unpath(/books/book[5] )

Xdmp:unpath(/books/book[6])

**Output>**

<book id="0">xml<authorname auid="101">Holmen</authorname><price>100rs</price></book>

<book id="1">xpath</book>

<book id="2">xquery</book>

<book id="3">core java</book>

<book id="4">j2ee</book>

<book id="5">struts 2.x</book>

1. **Function for properties:**

**Properties:**

1. **xdmp:document-properties([$uri as xs:string\*]) as document-node()\***

or

xdmp:document-properties()

E.g.:

xdmp:document-properties("/mydoc.xml") ,

xdmp:document-properties()

*=>*Returns a sequence of properties documents, one for each of the specified documents that has a corresponding properties document. If no documents are specified, returns a sequence of properties documents for all documents in the database that have a corresponding properties document.

*<prop:properties xmlns:prop="http://marklogic.com/xdmp/property">*

*<Win32FileAttributes xmlns="urn:schemas-microsoft-com:">00000020</Win32FileAttributes>*

*<Win32LastAccessTime xmlns="urn:schemas-microsoft-com:">Tue, 28 May 2013 13:15:36 GMT</Win32LastAccessTime>*

*<Win32LastModifiedTime xmlns="urn:schemas-microsoft-com:">Tue, 28 May 2013 13:15:36 GMT</Win32LastModifiedTime>*

*<prop:last-modified>2013-05-28T19:04:46+05:30</prop:last-modified>*

*</prop:properties>*

**Example:** xdmp:document-get-properties("/books.xml",xs:QName("prop:last-modified"))

**Output>** <prop:last-modified xmlns:prop="http://marklogic.com/xdmp/property">2013-05-20T15:45:01+05:30</prop:last-modified>

1. **cts:properties-query($query as cts:query) as cts:properties-query**

=>Returns a query that matches all documents where $query matches document-properties. When searching documents or document-locks, cts:properties-query provides a convenient way to additionally constrain the search against document-properties fragments.

**Example:**

cts:search(fn:collection(),cts:properties-query(cts:element-range-query(

xs:QName("prop:last-modified"),">",current-dateTime() - xs:dayTimeDuration("P1D"))))

=> All documents modified up to one day in the past.

Note that this example requires a dateTime range index on:

namespace: http://marklogic.com/xdmp/property

local name: last-modified

**Note: Range indexes are required for this query, and must be used with cts:search.**