What's New in MarkLogic 11

MarkLogic 11

Publication date 2023-10-16

Table of Contents

1. What's New in MarkLogic 11

MarkLogic 11 is a major release of MarkLogic Server that includes many new features and improvements.

Version Numbering

Beginning with version 11.0.0, MarkLogic Server is adopting the industry-standard Semantic Versioning number system. From this point forward, all public releases will use a 3-part version string, where the parts represent, respectively, MAJOR, MINOR, and PATCH releases, with respect to compatibility of public APIs.

- A release that increments only the PATCH version indicates that incorrect behaviors, what are considered "bugs", have been fixed or that security patches have been applied, but no compatibility changes are present
- A release that increments the MINOR version indicates that additional functionality has been included, but existing public APIs remain compatible
- A release that increments the MAJOR version indicates that additional functionality has been included and there are potentially breaking changes to one or more public APIs

Feature Highlights

- Optic Geospatial
- Optic Update (Technical Preview)
- Adaptive Memory Algorithms for Optic Sorts and Joins
- GraphQL
- New Look and Feel and Improved Accessibility
- New Monitoring and HA Capabilities

For a description of these and many more new features, see New Features in MarkLogic 11.

Bug Fixes

MarkLogic 11 includes fixes for a number of existing bugs. For details, see the Release Notes.

Known Incompatibilities

When you upgrade from MarkLogic 9 or 10, some of your applications will require minor changes to run correctly on MarkLogic 11. For details, see Known Incompatibilities with Previous Releases.

2. Release Notes

2.1. MarkLogic 11.1.0

MarkLogic 11.1.0 is a minor release that adds a number of new features and addresses a number of bugs and security vulnerabilities highlighted here:

- Automatic cache sizing will now allocate caches using the recommended ratios for hosts with up to 512 GB of RAM. Additional testing and tuning will be required to increase automatic calculation for hosts with more that 512 GB of RAM in later releases.
- Starting in MarkLogic 11.0.0, the Optic/SPARQL query plan viewer in Query Console shows the memory component of each step in the plan as "NaN". This is fixed in 11.1.0.
- Fixed an issue where `xdmp:logfile-scan()` might only return part of the lines for log entries that have many lines.
- Fixed an issue in the Admin Interface that causes replica databases when database encryption is enabled to report that the encrypted size is zero.
- Fixed an issue in the Admin Interface that prevented the modules database from being set for the triggers in a database.
- Fixed an issue in Query Console where Select all via [ctrl]+[A] was working incorrectly when selecting in the query buffer or results panel.
- Fixed an issue in Query Console that prevented copying and pasting the documentation from the autocomplete pop-up.
- Added support for backups to S3 buckets with object lock and compliance retention activated.
- Added support for a new "extension" option to `xdmp:document-filter()` and `xdmp.documentFilter()` that fixes an issue where the file type may not be detected correctly because the function does not know the file extension which otherwise comes from the document URI.
- Fixed multiple issues with the segment assignment policy:
  - Fixed the issue that causes potential data loss when using segment policy and retired forests.
  - Fixed the issue that causes potential data loss when rebalancing with the segment policy.
  - Fixed the issue that causes an increase of deadlocks during rebalancing.
- Improved error messages for XDMP-LEXVAL errors.
- Improved error handling when too many values are passed into calls to `cts:triple-range-query()`.
- Fixed performance regression starting from 10.0-9 where an optimization for certain `cts:triples()` patterns that were added caused other triple range queries with `cts:triples()` queries to be slower.
- The metrics for "external" join and sort are now available via the REST API `/manage/v2/meters` GET endpoint.
- Added missing support for the `cts.documentFormatQuery()`, `cts.documentRootQuery()`, and `cts.documentPermissionQuery()` query constructors to Optic DSL.
- Upgraded the libraries in XCC and mlcp to address multiple security vulnerabilities

NOTICE

Due to the Java version requirements of the updgraded 3rd party libraries, mlcp now requires JRE 11 or later.

  - CWE-400
  - CWE-611
  - CVE-2023-2976

  - CVE-2020-8908
  - CVE-2022-45688
  - CVE-2007-1157
  - CWE-190
- There is a known issue with mlcp loading compressed RDF files from zip archives where the individual RDF file is over 2 GB. This will be fixed in the next release.

To review the full list of bug fixes included MarkLogic 11.1.0, visit the BugTrack page on the MarkLogic Support Portal. Select the "From" version you would like to start with and set the "To" version to "11.1.0".

2.2. MarkLogic 11.0.3

MarkLogic 11.0.3 is a patch release that addresses a number of bugs and contains some performance improvements. The following are some of the highlights:

- op.fromSearch() and op.fromSearchDocs() security constraints
  - Starting with MarkLogic 10.0-6, Optic queries using either op.fromSearch()/op:from-search() or op.fromSearchDocs()/op:from-search-docs() operators may return more documents than theoretically accessible. This has been fixed in 11.0.3 as well as 10.0-6.6, 10.0-7.4, 10.0-8.5, 10.0-9.7, and 10.0-10.2.
- The v1/search REST API with the "return-constraints" query option and JSON format now returns valid JSON even if there are multiple constraints
- Fixed an issue where segfaults can happen when performing rolling upgrades to 11.0.0 on multiple clusters that have been coupled for database replication
- Element Level Security fix
  - If multiple protected paths are protecting the same elements with different attribute values, some protected elements fail to be protected while some other protected elements fail to show up to users who should have access
- Geospatial triples of type LINESTRING or POLYGON are now deduplicated correctly in SPARQL results
- Performance improvement of region relate queries such as 'CONTAINS', 'WITHIN', 'CROSSES', 'OVERLAPS', 'EQUALS', 'DISJOINT', 'INTERSECTS', 'TOUCHES', 'COVERS', and 'COVERED BY' that are run while updates are occurring
- Fixed an issue where users might encounter the error `curlCode: 28, Timeout was reached` when doing backups to s3 in very large clusters
- Improved performance of SQL or Optic queries against sys_columns with query-based views
- In a AWS Managed Cluster, EBS volumes are now mounted with 'nodev' flag and the /var/tmp/marklogic.host file is written with permissions of rw-rw-rw-
- Fixed an issue where the server hangs when turning log encryption on when telemetry is enabled
- Connections to AWS KMS, Azure KeyVault and other key management systems now support TLSv1.2
- Fixed an issue where server-side JavaScript code referencing XQuery modules with the same name may have used the wrong module
- The "property" query string parameter is now supported in the v2/forests REST API
- Fixed multiple security vulnerabilities in XCC and mlcp:
  - CVE-2007-1157
  - CVE-2022-45688

To review the full list of bug fixes included MarkLogic 11.0.3, visit the BugTrack page on the MarkLogic Support Portal. Select the "From" version you would like to start with and set the "To" version to "11.0.3".

2.3. MarkLogic 11.0.2

MarkLogic 11.0.2 is a patch release that addresses a number of bugs in MarkLogic 11.0.0. The following are some of the highlights:

- Admin UI improvements
  - Performance improvements to the status pages for large clusters or databases with hundreds of forests
  - Improved form input validation for certificate templates
  - Report when there are hosts in the cluster on different patch versions
- Performance improvements for Optic `fromSearchDocs()` and `joinDocCols()`
- Performance improvements for starting backups of databases with a large number of forests
- The new health check REST endpoint is now configured correctly when upgrading from an earlier version
- Performance and stability improvements for backups to S3
- Addressed an error in PDF conversion
- Fixed `op.xpath()` support for XPaths that start with `./` or `../`
- Addressed a SQL issue where `LEFT JOIN` with `IS NULL` condition on a nullable column unexpectedly returned no results
- Addressed a SPARQL issue where certain queries with `OPTIONAL` clauses returned the wrong results
- Added support for new AWS EC2 instance types
- Addressed an issue with scheduling of weekly backups
- Addressed an issue with serialization of CTS queries for Query-Based Views

To review the full list of bug fixes included MarkLogic 11.0.2, visit the BugTrack page on the MarkLogic Support Portal. Select the "From" version you would like to start with and set the "To" version to "11.0.2".

2.4. MarkLogic 11.0.0

MarkLogic 11.0.0 includes fixes for a number of existing bugs. The following are some of the highlights:

- mlcp now handles splits with multi-byte content
- AWS Managed Cluster capability now better handle EBS volume and ENI reattachment for failed instances
- Reindexing is now triggered automatically when creating range indexes that use document properties
- Search results are now correct for JSON documents with "lang" or "language" property
- Libraries have been upgraded to address security vulnerabilities

To review the full list of bug fixes included MarkLogic 11.0.0, visit the BugTrack page on the MarkLogic Support Portal. Select the "From" version you would like to start with and set the "To" version to "11.0.0".

3. Installation and Upgrade

This section describes the supported platforms and upgrade paths for MarkLogic Server.

3.1. Supported Platforms

For a complete list of supported platforms, see Supported Platforms in *Installing MarkLogic Server*.

3.2. Supported Filesystems

For a complete list of supported filesystems, see Supported Filesystems in *Installing MarkLogic Server*.

3.3. Upgrade Support

This section describes upgrade support to MarkLogic 11. For details on installing MarkLogic Server and for the upgrade procedure, see *Installing MarkLogic Server*.

WARNING

MarkLogic Early Access does not support upgrades. This section describes upgrading from 9.0-1 and later.

IMPORTANT

MarkLogic Corporation strongly recommends performing a backup of your databases before upgrading to MarkLogic 11. Additionally, MarkLogic Corporation recommends that you first upgrade to the latest maintenance release of the major version of MarkLogic you are running before upgrading to MarkLogic 11.

An upgrade from MarkLogic 9 or 10 does not require a reindex.

IMPORTANT

If you are upgrading clusters with DB replication configured, see Upgrading Clusters Configured with Database Replication in the *Database Replication Guide*.

There are some known incompatibilities between MarkLogic 9 and MarkLogic 11. You might need to make some minor code changes to your MarkLogic 9 applications before they can run correctly in MarkLogic 11. For details on the incompatibilities, see Known Incompatibilities with Previous Releases. For instructions on upgrading to MarkLogic 11, including information about database compatibility between MarkLogic 9 and MarkLogic 11, see *Installing MarkLogic Server*.

4. New Features in MarkLogic 11.1.0

This section describes the new features in MarkLogic 11.1.0.

4.1. XQuery and Node.js Support for Optic Update (Technical Preview)

> **IMPORTANT**
> Technical Preview means a feature that is in whole or in part under active development and in stages of testing. Its purpose is to establish a feedback loop, allowing customers to influence its development and direction. Backward-incompatible changes may be introduced to the service or its APIs.

MarkLogic 11.1.0 adds XQuery support for the Optic Update feature that was introduced in 11.0.0.

This capability is released as a Technical Preview (defined above) with the intention of removing the known limitations and incorporating user feedback in coming minor releases of MarkLogic.

New Operators

Optic Update includes three new Optic operators for performing updates:

Function	Description
op:lock-for-update()	Operator that gets an early lock on documents that will be updated later in the pipeline with an operation like `remove()` or `write()`.
op:remove()	Operator that deletes documents.
op:write()	Operator that inserts or overwrites documents as supplied by document descriptors.

New operators have also been added to support common scenarios when updating documents. These operators will often be used with update operators, but they can be used with existing Optic query operators as well:

Function	Description
op:from-doc-descriptors()	Data accessor that returns document rows from one or more "document descriptors," which are a combination of URI, the document, collections, metadata, permissions, quality, and temporal collection.
op:from-doc-uris()	Data accessor that returns a list of URIs that match a given CTS query.
op:from-param()	Data accessor that returns document rows from a given set of parameters.
op:join-doc-cols()	Operator that can be used to join in document descriptor columns using the supporting `op:doc-cols()` and `op:doc-col-types()` functions.
op:transform-doc()	Operator that applies a transformation to the documents in the document column for each row.
op:validate-doc()	Operator that validates documents based on a supplied JSON, XML, or Schematron schema.
op:execute()	Executor that executes the Optic pipeline/plan but does not return results out of the pipeline. This is useful if you want to update a list of documents but not actually return anything from the pipeline.

Examples

This example does a simple insert of a single document, specifying the collections as well:

```
xquery version "1.0-ml";
import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";
declare option xdmp:update "true";

let $doc-descriptors := (
    map:entry("uri", '/helloworld.xml')
      =>map:with("doc", <doc>hello world</doc>)
      =>map:with("collections", ("mydocs"))
  )
return op:from-doc-descriptors($doc-descriptors)
=>op:write()
=>op:execute()
```

`op:validate-doc()` can be used to validate documents against a schema as they are inserted. This example validates the given document against the schemas installed. In this example, `op:select()` is used to just return the URI and `op:result()` is used instead of `op:execute()` so that we can see which URIs were inserted vs. had validation errors. Validation errors are currently reported in the app server error log:

```
xquery version "1.0-ml";
import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";
declare option xdmp:update "true";

let $doc-descriptors := (
    map:entry("uri", '/helloworld.xml')
      =>map:with("doc", <doc xmlns="http://marklogic.com/xdmp/helloWorld">hello world</
doc>)
      =>map:with("collections", ("mydocs"))
  )

return op:from-doc-descriptors($doc-descriptors)
=>op:validate-doc(op:col("doc"), map:entry("kind", "xmlSchema")
=>map:with("mode", "strict"))
=>op:write()
=>op:select("uri")
=>op:result()
```

`op:remove()` can be used to delete a list of documents. Once again, `op:result()` can be used to return the list of URIs that were removed. `op:execute()` can be used if the result list is not needed:

```
xquery version "1.0-ml";
import module namespace op="http://marklogic.com/optic"
     at "/MarkLogic/optic.xqy";
declare option xdmp:update "true";

op:from-doc-uris(cts:word-query("world", ("case-sensitive")))
=>op:remove()
=>op:result();
```

Node.js Client

As of version 3.1.0, the MarkLogic Node.js client also supports the use of the Optic Update operators.

Limitations

- The "patch" operator that supports declarative modifications of documents is not yet available.
- The error handling operator that allows detailed control of error disposition is not yet available.

Additional Information

- *Getting Started with Optic*

- Optic Update Functions
- Optic Data Access Functions
- Optic Operator Functions
- Optic Executor Functions
- Optic API for Multi-Model Data Access Guide

4.2. Reverse Proxy and Load Balancer Path-Based Routing

Server User Interfaces

As of MarkLogic 11.1.0, the MarkLogic Admin, Query Console, and Monitoring Interfaces can now be accessed using path-based routing through a reverse proxy or load balancer. For example, a reverse proxy could be configured to route all requests to the "marklogic/admin" path to port 8001 of the MarkLogic hosts running behind the proxy.

To configure a reverse proxy or load balancer, see Running Behind a Load Balancer or Reverse Proxy in *Administrating MarkLogic Server*.

NOTICE

Digest and certificate authentication are not currently supported when running behind a reverse proxy or load balancer configured with path-based routing. Configure the MarkLogic app servers to use either basic auth over HTTPS or one of the other supported authentication mechanisms.

mlcp

As of MarkLogic 11.1.0, mlcp supports accessing MarkLogic through a reverse proxy or load balancer using path-based routing.

mlcp now supports the `-base_path` command line option that can be specified when connecting to MarkLogic. This would be a path that is configured on the reverse proxy or load balancer to route to an XDBC app server port of the MarkLogic hosts running behind the proxy or load balancer.

For more details, see mlcp Reverse Proxy Support Using Path-Based Routing in *mlcp User Guide*.

XCC

As of MarkLogic 11.1.0, XCC supports the `basePath` parameter when connecting to a MarkLogic XDBC app server. For more details, see XCC Reverse Proxy Support in *Developing with XCC*.

Java Client

Starting with version 6.1.0, the Java Client supports the `basePath` parameter when connecting to MarkLogic. For more information, see the Javadocs for the DatabaseClientBuilder.

Node.js Client

Starting with version 3.0.0, the Node.js Client supports the `basePath` parameter when connecting to MarkLogic. For an example that uses this parameter, see the MarkLogic Cloud connection example.

4.3. Forest Memory Diagnostics

MarkLogic 11.1.0 adds the ability to retrieve detailed memory usage metrics at the forest level. When either the "memory-summary" or "memory-detail" option is provided to `xdmp.forestStatus()`, new metrics about index memory usage are returned. These metrics can be used to better understand the amount of memory that each range index or lexicon are using.

The result from a call to `xdmp.forestStatus()` with the "memory-detail" option includes an ID of each index. In order to map the index IDs to actual indexes configured in the database, a new function `xdmp.databaseDescribeIndexes()` has been added in MarkLogic 11.1.0.

In addition to the new built-in functions in JavaScript and XQuery, the memory diagnostics are also available via the REST API. The `/manage/v2/forests/{id|name}?view=status` GET endpoint now supports the `property` URI parameter that can be set to either "memory-summary" or "memory-detail" to retrieve the forest-level memory metrics.

A new "describe-indexes" view has been added to the databases GET endpoint that can be used to retrieve the mapping from index IDs to indexes configured in the database.

4.4. PKI and KMS Auditing

MarkLogic 11.1.0 adds the ability to turn on additional audit events for the Public Key Infrastructure (PKI) and Key Management System (KMS) levels. Two new audit events have been added to support this:

- **PKI-user** - Audits when any public API operations occur that are related to PKI or KMS usage.
- **PKI-system** - Audits when any internal server operations occur that are related to PKI or KMS usage.

For the full list of auditable events, see Auditable Events in *Administrating MarkLogic Server*.

Additionally, MarkLogic 11.1.0 adds more granular control of encryption of log files. Starting with 11.1.0, the audit log files can be encrypted independent from the access, error and request logs. This is a new cluster-level setting. For details, see Encrypting Data, Configuration, and Log Files in *Administrating MarkLogic Server*.

4.5. New Full-Width UI Banner

In MarkLogic 11, a configurable label that appears in the header that is common across all the built-in applications was added. In 11.1.0, support for an addition full-width "banner" was added. The full-width banner is configurable via the "topBannerLabel" and "topBannerStyle" properties that can be set via the `admin.uiSetBanner()` built-in function.

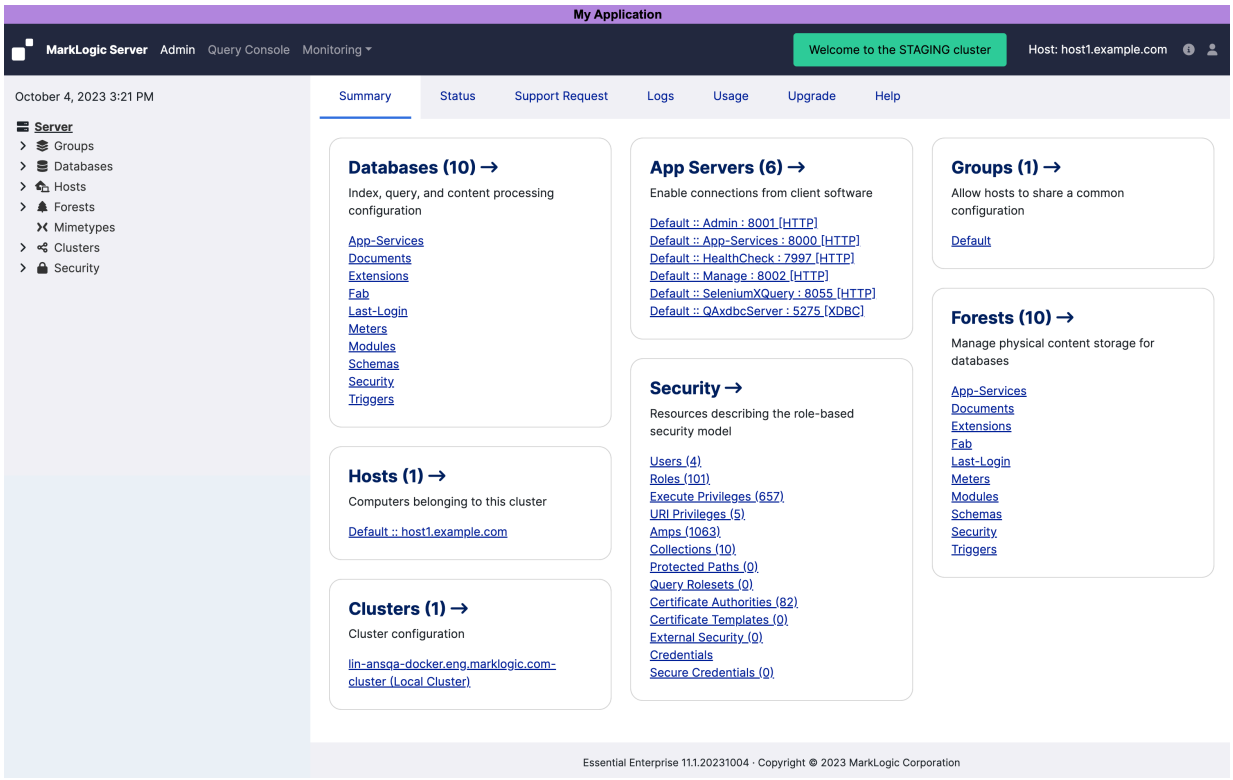
The "topBannerStyle" is an HTML style that can contain CSS properties that control the style of the full-width banner. For example, the following can be run in Query Console to configure a full-width banner as well as the header banner:

```
'use strict';
const admin = require("/MarkLogic/admin.xqy");

let uiSetting = {
  "active": true,
  "label": "Welcome to the STAGING cluster",
  "headerColor": "#33CC99",
  "headerTextColor": "#000000",
  "message": null,
  "topBannerLabel": "My Application",
  "topBannerStyle": "background-color:#b587de;color:#000000;font-family:arial;font-
size:16px;font-weight:bold;text-align:center;"
}

admin.uiSetBanner(uiSetting)
```

This sets the top banner and header banner so that they appear as shown in the following image:



Known Issues

Under some conditions, when the top banner is visible, the menu header will overlap with the tabs in Query Console, making the tabs inaccessible. A slight resizing of the browser window or the Query Console buffer fixes the issue. This issue is currently under investigation, and a fix will be made available as soon as one can be identified and scheduled for release.

4.6. AWS CloudFormation Template (CFT) Improvements

IMDSv2 support

MarkLogic 11.1.0 adds support for version 2 of the EC2 Instance Metadata Service (IMDSv2). This is the latest and most secure option for managing and accessing EC2 instance metadata and is now the default when launching MarkLogic from version 11.1.0 of the CloudFormation Templates. The MarkLogic AMIs still default to IMDSv2 for backward compatibility but the IMDSv2 option is set to "required" by default in the 11.1.0 and later CFTs.

In order to use MarkLogic Server AMIs before 11.1.0 with the new templates, the templates will need to be modified to set IMDSv2 to "optional" as IMDSv2 is not supported in earlier versions of the MarkLogic AMI. See the AWS Security Blog for more details about IMDSv2.

Launch templates

Starting with MarkLogic 11.1.0, the MarkLogic CloudFormation Templates replace the use of Launch Configurations with Launch Templates. This ensures that MarkLogic CFT users can make use of all of new EC2 features now available in AWS Launch Templates. See the AWS Compute Blog for more details about the introduction of AWS Launch Templates.

NOTICE

The use of Launch Templates in the CFTs requires that new privileges be added to the the IAM role used to launch the CloudFormation stacks. Add the following privileges to the IAM role used to launch MarkLogic clusters via the CFTs:

- "ec2:CreateLaunchTemplate"
- "ec2:DescribeLaunchTemplates"
- "ec2:DeleteLaunchTemplate"
- "ec2:ModifyLaunchTemplate"
- "Resource": "arn:aws:ec2:::launch-template/*"

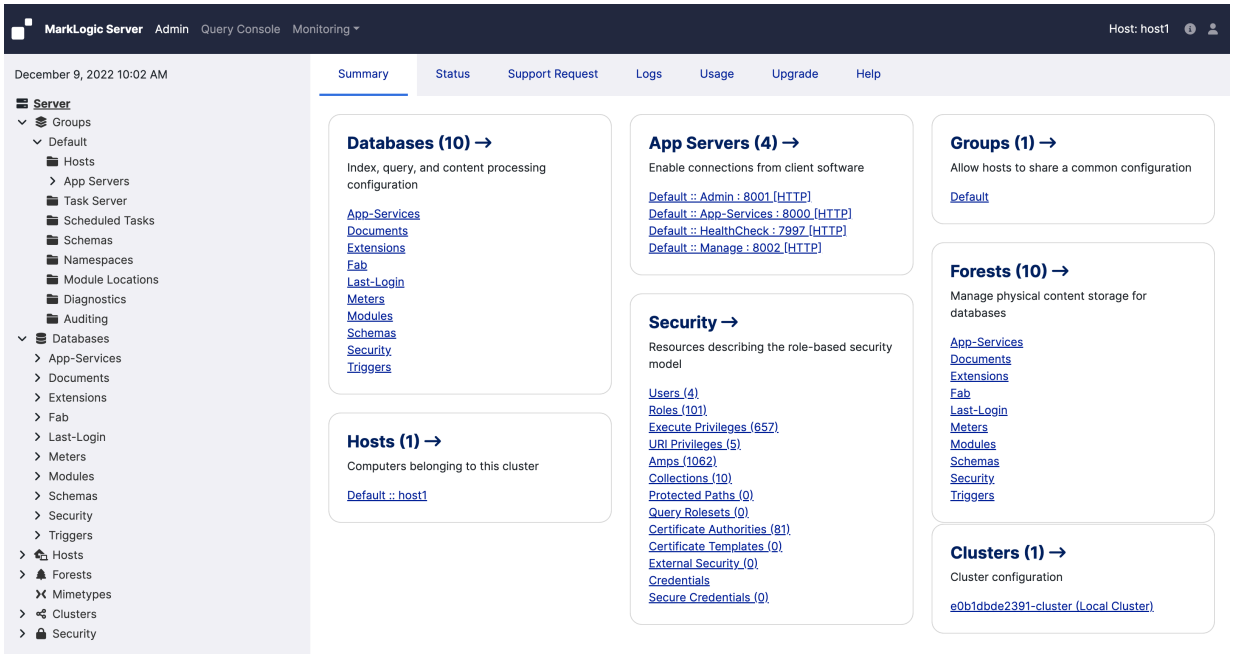
See Creating an IAM Role in *Getting Started with MarkLogic Server on AWS* for the complete list of additional privileges required.

5. New Features in MarkLogic 11

This section describes the new features in MarkLogic 11.

5.1. New Look and Feel and Improved Accessibility

MarkLogic 11 introduces a new look and feel for the Admin UI which unifies our application branding with a common header and style guide across all the built-in applications. The new Admin UI focuses on accessibility and compliance, brings a more intuitive tree navigation, and introduces responsive design. This is a first step in modernizing the look and feel of all of the MarkLogic applications, so expect to see more improvements in coming releases.



Accessibility and 508 Compliance

In addition to the new look and feel, the applications have been updated to improve accessibility and have been evaluated for 508 compliance. The applications were evaluated by a 3rd-party firm, and the Voluntary Product Accessibility Template (VPAT) will be available for download from www.marklogic.com/trust once it has been published.

Configuring the Banner

In MarkLogic 11, a banner and notification message that is common across all the built-in applications can be configured via new built-in functions.

Function	Description
admin.uiGetBanner()	Retrieves the current banner configuration
admin.uiSetBanner()	Sets whether the banner is active, the label, colors, and message to display when users first log in

5.2. GraphQL

MarkLogic 11 provides a GraphQL query service with out-of-the-box support for querying user-defined views. Those views may be defined via Template Driven Extraction or Query-Based Views. Additionally, the views and the relationships between the views may be augmented based on a custom GraphQL schema. This service is accessible by sending a JSON object with a "query" property containing a GraphQL query string to the /v1/rows/graphql REST endpoint using either an HTTP GET or HTTP POST request. Those requests must set the Content-Type header to "application/graphql".

Features

The GraphQL query service supports the following features:

Feature	Description
View Arguments	This version supports arguments for specifying a value to filter a column on. See https://graphql.org/learn/queries/#arguments.
Range Directives	These are a set of directives (ie. @greaterThan) that may be used to specify a range filter for a column.
Sort Directive	This directive allows for specifying a sort column and a direction for the resulting records.
Pagination Arguments	The "first" and "offset" arguments allow for pagination. These arguments should only be used in the root view of a query.
Implicit GraphQL Schemas	Implicit GraphQL schemas are generated by MarkLogic at request time to reflect the user-defined views available in the database.
Explicit GraphQL Schemas	These user-defined schemas are used to augment or customize the default schema.
Join Directive	This directive is used in explicit schemas to relate two views and to define the join columns for that relationship.
ID Directive	This directive is used in explicit schemas to define the primary key column for a view. That primary key column is then used for object grouping in the JSON responses.
View & Field Aliases	Aliases may be used to rename views and columns in the resulting JSON objects. See https://graphql.org/learn/queries/#aliases.
Multiple Queries in a single operation	A single query operation may contain multiple queries. The results are included in a single JSON response object.

REST APIs

The following REST APIs are available in MarkLogic 11:

API	Description
/REST/POST/v1/rows/graphql	Executes a GraphQL query in order to retrieve a JSON object with the results. The query is passed in the "query" property of the JSON request body.
/REST/GET/v1/rows/graphql	Executes a GraphQL query in order to retrieve a JSON object with the results. The query parameter of the GET URL should be JSON containing a "query" property.

GraphQL Schemas

Implicit GraphQL schemas are automatically generated by MarkLogic during request processing to reflect the user-defined views available in the database. They define the default queries and data types that are available to users of the service and are user-specific.

Explicit GraphQL schemas are created by a MarkLogic administrator in order to augment or customize the implicit GraphQL schema. They can be used for defining joins and/or limiting columns available in views. While processing a request, if an explicit GraphQL schema has been defined and the GraphQL service has been configured to use that schema, then the service merges the implicit GraphQL schema with the explicit GraphQL schema. Any types or queries that are defined in both schemas are overwritten with the version in the explicit schema.

Limitations

This initial version of the GraphQL service is focused on basic query handling. As such, non-query operations and some optional query features are not yet available.

- Mutations - Users will not be able to send updates to the database via GraphQL.
- Subscriptions - Users will not be able to subscribe to queries for receiving data pushes from the server.
- Introspection - Users will not be able to request schema information from the server using GraphQL. However, it is possible to get schema information using QConsole.
- Multi-Operation Requests - The service will only provide a response to the first query operation in a request. Any additional query operations will be ignored.

- Skip Directive - Users will not be able to use this directive to specify columns to exclude under specific conditions.
- Include Directive - Users will not be able use this directive to specify columns to include under specific conditions.
- Variables - Users will not be able to define or use variables in queries.
- Fragment - Users will not be able to define or use fragments of queries.
- Custom Grouping and Aggregation - Users will not be able to set custom grouping or aggregations as part of a query. However, a work-around is to implement a query-based view with the desired grouping and aggregation in the MarkLogic server and use that query.
- Custom Resolvers - Users will not be able to customize how views and columns are resolved.

Known Issues

- Using the "limit" or "offset" arguments on joined views does not work as expected for GraphQL. The pagination takes place for the entire joined view; therefore, the result set may not have all the expected data for the joined view.
- Joins can lead to very large result sets. Pagination on the root view only affects the number of root view objects returned. Therefore, those root view objects may have a very large number of child objects.
- Empty object arrays in 3rd-tier joins have single objects with null values. It is recommended that the user check child objects for null values.

5.3. Optic Geospatial

In MarkLogic 11, geospatial data can now be indexed using Template Driven Extraction and queried using the Optic engine, SPARQL, and SQL. This builds on the the rich geospatial capabilities already built into MarkLogic, and adds flexibility to how geospatial indexes are built and queried. It also increases interoperability by implementing subsets of the OGC (Open Geospatial Consortium) standard for SQL queries and OGC (Open Geospatial Consortium) GeoSPARQL standard for SPARQL queries.

TDE Indexing

The TDE "scalar-type" has been extended to support the following new scalar types: `point` , `box` , `circle` , `linestring` , `polygon` , `complexPolygon` , and a generic `region`. Using these types, geospatial data can be selected for indexing using standard TDE XPath.

The TDE value expression must extract and/or construct content into one of the formats currently supported by the MarkLogic geospatial region index. See Geospatial Region Queries and Indexes in the *Search Developer's Guide* for more details.

The following code excerpt shows a fragment of a TDE that creates a row for each building and selects the building boundary polygon from the "boundary" element or property:

```
"rows": [
  {
    "schemaName": "City",
    "viewName": "Buildings",
    "columns": [
      {
        "name": "name",
        "scalarType": "string",
        "val": "name"
      },
      {
        "name": "boundary",
        "scalarType": "polygon",
        "val": "cts:polygon(boundary)",
        "coordinateSystem": "wgs84"
       }
     ]
  }
]
```

See Template Driven Extraction in the *Application Developer's Guide* for more details on how to to use TDE.

Optic Query

Using the view from the above template, the Optic API can be used to query for buildings that intersect a circle with a center point and radius:

```
const op = require('/MarkLogic/optic');
op.fromView('City', 'Buildings')
  .where(
    op.geo.intersects(
      op.col('boundary'),
      geo.circlePolygon(cts.circle(10, cts.point(20.7, -156.4)), 0.5)
    )
  )
  .result()
```

The following functions have been added and are available under the existing "geo" namespace:

Function	Returns
`op.geo.crosses(region1 as cts.region, region2 as cts.region)`	`true` if `region1` crosses `region2`, `false` otherwise
`op.geo.within(region1 as cts.region, region2 as cts.region)`	`true` if `region1` is within `region2`, `false` otherwise
`op.geo.contains(region1 as cts.region, region2 as cts.region)`	`true` if `region1` contains `region2`, `false` otherwise
`op.geo.overlaps(region1 as cts.region, region2 as cts.region)`	`true` if `region1` overlaps `region2`, `false` otherwise
`op.geo.equals(region1 as cts.region, region2 as cts.region)`	`true` if `region1` equals `region2`, `false` otherwise
`op.geo.disjoint(region1 as cts.region, region2 as cts.region)`	`true` if `region1` is disjoint from `region2`, `false` otherwise
`op.geo.intersects(region1 as cts.region, region2 as cts.region)`	`true` if `region1` intersects `region2`, `false` otherwise
`op.geo.touches(region1 as cts.region, region2 as cts.region)`	`true` if `region1` touches `region2`, `false` otherwise
`op.geo.covers(region1 as cts.region, region2 as cts.region)`	`true` if `region1` covers `region2`, `false` otherwise
`op.geo.coveredBy(region1 as cts.region, region2 as cts.region)`	`true` if `region1` is covered by `region2`, `false` otherwise

SQL Query

Using the view from the above template, SQL can be used to query for buildings that intersect a circle with a center point and radius:

```
select * from Buildings where ST_Intersects(
  boundary,
  geo_circle_polygon(cts_circle(10, cts_point(20.7, -156.4)), 0.5)
)
```

The following functions from the OGC standard for SQL queries are supported in MarkLogic 11:

Function	Example SQL	Returns
ST_WKTToSQL()	select ST_WKTToSQL('POLYGON((0.39 -.10, 0.42 1.4, 0.5 2.0, 0.39 -0.10))');	cts:region()
ST_WKBToSQL()	select ST_WKBToSQL(geo_to_wkb('POINT(0.39 -0.10)'));	cts:region()
ST_AsText()	select ST_AsText(cts_point(0.39, -0.10));	string
ST_AsBinary()	select ST_AsBinary(cts_point(0.39, -0.10));	binary
ST_Crosses()	select * from polygons WHERE ST_Crosses(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Within()	select * from polygons WHERE ST_Within(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Contains()	select * from polygons WHERE ST_Contains(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Overlaps()	select * from polygons WHERE ST_Overlaps(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Equals()	select * from polygons WHERE ST_Equals(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Disjoint()	select * from polygons WHERE ST_Disjoint(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Intersects()	select * from polygons WHERE ST_Intersects(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Touches()	select * from polygons WHERE ST_Touches(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Covers()	select * from polygons WHERE ST_Covers(cts_point(0.39, -0.10), polygon_column)	boolean
ST_CoveredBy()	select * from polygons WHERE ST_CoveredBy(cts_point(0.39, -0.10), polygon_column)	boolean
ST_Distance()	select ST_Distance(ST_WKTToSQL('POINT (30 10)'), point_column) from points	double
ST_Centroid()	select ST_Centroid(ST_WKTToSQL('POLYGON((0.39 -.10, 0.42 1.4, 0.5 2.0, 0.39 -0.10))'));	cts:point()
ST_PointOnSurface()	select ST_PointOnSurface(ST_WKTToSQL('POLYGON((0.39 -.10, 0.42 1.4, 0.5 2.0, 0.39 -0.10))'))	cts:point()

SPARQL Query

The following functions from the OGC (Open Geospatial Consortium) GeoSPARQL standard for SPARQL queries are supported in MarkLogic 11. These all use the SPARQL prefix: `PREFIX geof: <http://www.opengis.net/def/function/geosparql/>`:

Function	Returns
`geof:sfCrosses($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` crosses `$region2`, `false` otherwise
`geof:sfWithin($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` is within `$region2`, `false` otherwise
`geof:sfContains($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` contains `$region2`, `false` otherwise
`geof:sfOverlaps($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` overlaps `$region2`, `false` otherwise
`geof:sfEquals($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` equals `$region2`, `false` otherwise
`geof:sfDisjoint($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` is disjoint from `$region2`, `false` otherwise
`geof:sfIntersects($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` intersects `$region2`, `false` otherwise
`geof:sfTouches($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` touches `$region2`, `false` otherwise
`geof:sfCovers($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` covers `$region2`, `false` otherwise
`geof:sfCoveredBy($region1 as cts:region(), $region2 as cts:region(), $options as xs:string*)`	`true` if `$region1` is covered by `$region2`, `false` otherwise
`geof:distance($point1 as cts:point(), $point2 as cts:point(), $options as xs:string*)`	The distance, in the units specified in `$options` (miles by default), between `$point1` and `$point2`

The following functions are new built-in functions supported in MarkLogic 11:

Function	Description
geo.within()	Compares geospatial regions to see if they fulfill the 'within' DE-9IM relation.
geo.contains()	Compares geospatial regions to see if they fulfill the 'contains' DE-9IM relation.
geo.overlaps()	Compares geospatial regions to see if they fulfill the 'overlaps' DE-9IM relation.
geo.equals()	Compares geospatial regions to see if they fulfill the 'equals' DE-9IM relation.
geo.disjoint()	Compares geospatial regions to see if they fulfill the 'disjoint' DE-9IM relation.
geo.intersects()	Compares geospatial regions to see if they fulfill the 'intersects' DE-9IM relation.
geo.touches()	Compares geospatial regions to see if they fulfill the 'touches' DE-9IM relation.
geo.covers()	Compares geospatial regions to see if they fulfill the 'covers' DE-9IM relation.
geo.coveredBy()	Compares geospatial regions to see if they fulfill the 'covered by' DE-9IM relation.

Additional Information

See Query Geospatial Data in Optic for a detailed walkthrough of working with the new Optic Geospatial capability.

See *Getting Started with Optic* for more detail about working with the Optic API.

See Geospatial Search Applications in the *Search Developer's Guide* for more detail about working with geospatial data in MarkLogic.

5.4. Optic Update (Technical Preview)

IMPORTANT

Technical Preview means a feature that is in whole or in part under active development and in stages of testing. Its purpose is to establish a feedback loop, allowing customers to influence its development and direction. Backward-incompatible changes may be introduced to the service or its APIs.

MarkLogic 11 adds a new capability that allows developers to perform inserts, updates, and deletes via the Optic API. This expands the capability of MarkLogic's unique multi-model query language to make it easier to insert and update documents without having to write server-side code.

With the new Optic Update functions, developers can start an Optic pipeline with one or more new or existing documents as input, transform the documents, add metadata, and write them to the database in a single expression.

This capability is released as a Technical Preview (defined above) with the intention of removing the known limitations and incorporating user feedback in coming minor releases of MarkLogic.

New Operators

Optic Update includes three new Optic operators for performing updates:

Function	Description
op.lockForUpdate()	Operator that gets an early lock on documents that will be updated later in the pipeline with an operation like remove() or write()
op.remove()	Operator that deletes documents
op.write()	Operator that inserts or overwrites documents as supplied by document descriptors

New operators have also been added to support common scenarios when updating documents. These operators will often be used with update operators, but they can be used with existing Optic query operators as well:

Function	Description
op.fromDocDescriptors()	Data accessor that returns document rows from one or more "document descriptors" which are a combination of URI, the document, collections, metadata, permissions, quality, and temporal collection
op.fromDocUris()	Data accessor that returns a list of URIs that match a given CTS query
op.fromParam()	Data accessor that returns document rows from a given set of parameters
op.joinDocCols()	Operator that can be used to join in document descriptor columns using the supporting op.docCols() and op.docColTypes() functions
op.transformDoc()	Operator that applies a transformation to the documents in the document column for each row
op.validateDoc()	Operator that validates documents based on a supplied JSON, XML, or Schematron schema
op.execute()	Executor that executes the Optic pipeline/plan but does not return results out of the pipeline. This is useful if you want to update a list of documents but not actually return anything from the pipeline.

Examples

This first example just does a simple insert of a single document, specifying the collections as well.

```
declareUpdate();
const op = require("/MarkLogic/optic");
op
  .fromDocDescriptors({
    uri: "/helloworld.json", doc: {"hello": "world"}, collections: [ "mydocs" ]
  })
  .write()
  .execute()
```

op.validateDoc() can be used to validate documents against a schema as they are inserted. In this example, op.select() is used to just return the URI and op.result() is used instead of op.execute() so we can see which URIs were inserted vs. had validation errors. Validation errors are currently reported in the app server error log.

```
declareUpdate();
const op = require("/MarkLogic/optic");
op
  .fromDocDescriptors({
    uri: "/helloworld.json", doc: {"hello": "world"}, collections: [ "mydocs" ]
  })
  .validateDoc('doc', {kind:'jsonSchema', schemaUri:'/myschema.json'})
  .write()
  .select('uri')
  .result()
```

op.remove() can be used to delete a list of documents. Once again, op.result() can be used to return the list of URIs that were removed. op.execute() can be used if the result list is not needed.

```
declareUpdate();
const op = require("/MarkLogic/optic");
op
  .fromDocUris(cts.wordQuery('world'))
  .remove()
  .result()
```

Limitations

- The new Optic Update operators are only available in server-side JavaScript
- "Builders" for the new Optic Update operators have only been implemented in the Java Client
- The "patch" operator that supports declarative modifications of documents is not yet available
- The error handling operator that allows detailed control of error disposition is not yet available

Additional Information

See *Getting Started with Optic* for more detail about working with the Optic API.

Optic Update Functions

Optic Data Access Functions

Optic Operator Functions

Optic Executor Functions

Optic API for Multi-Model Data Access Guide

5.5. Adaptive Memory Algorithms for Optic Sorts and Joins

MarkLogic 11 adds the ability for large sorts and joins in the Optic engine to overflow to disk. This provides improved support for large analytic queries, reporting, and exports of large result sets. Prior to MarkLogic 11, these queries may have been canceled due to a lack of available RAM. In MarkLogic 11, the Optic engine has access to new algorithms for sorts and joins that limit the amount of RAM used and that overflow to a temporary location on disk when there is too much data to fit in RAM, allowing the query to complete.

Other than configuring a temporary directory if the default is not acceptable, there is nothing users need to do to take advantage of this new capability. The Optic query optimizer now has access to these algorithms and will select them when sufficiently large operations need to be performed or if memory is sufficiently constrained.

Temporary Files

The Optic engine encrypts all temporary files with continuously changing, random AES 256 keys. By default, the temporary files for large sorts and joins are written to the `Temp/Optic` directory under the default MarkLogic data directory. This will typically be `/var/opt/MarkLogic/Temp/Optic`.

For most use cases, there will be no reason to change this temporary directory. However, if there are cases where many large analytic queries are often overflowing to disk and the system could benefit from faster disks than are currently in use for the default data directory, in MarkLogic 11, the location of the `Temp` directory can be configured via the "temporary-directory" group setting (see /REST/PUT/manage/v2/groups/[id-or-name]/properties).

IMPORTANT

Because the temporary directory is a group-level setting, if it is set to a path other than the default, all hosts in the group must make a file system available to MarkLogic at that path. If MarkLogic cannot access the configured temporary directory, it will fall back to writing to the default location.

If, under special circumstances, the temporary directory needs to be overridden at the host level instead, the `MARKLOGIC_TEMP_DIR` environment variable can be set for the MarkLogic process. See the Best Practice Editing MarkLogic Server Environment Variables knowledge base article for more information about setting environment variables.

Diagnostics

In MarkLogic 11, xdmp.queryMeters() now reports per request the number of bytes read from temporary files and the number of bytes written to temporary files via `temporaryFileReadBytes` and `temporaryFileWriteBytes` respectively.

In MarkLogic 11, xdmp.serverStatus() now reports, on a per app server basis, the cumulative number of bytes read from temporary files and the cumulative number of bytes written to temporary files via `temporaryFileReadBytes` and `temporaryFileWriteBytes` respectively.

Additional Information

REST API to get group properties : /REST/GET/manage/v2/groups/[id-or-name]/properties

REST API to set group properties: /REST/PUT/manage/v2/groups/[id-or-name]/properties

See *Getting Started with Optic* for more detail about working with the Optic API.

5.6. Other Optic Improvements

op.fromLexicons() performance improvements

Significant performance improvements were made to the Optic Data Accessor op.fromLexicons(). For some use cases, this is now 10 to 100 times faster in MarkLogic 11.

cts.iriReference() to optimize joining documents to triples

A new reference for the URI lexicon called cts.iriReference() was added in MarkLogic 11 to improve performance when joining documents to triples. This should be used in place of cts.uriReference() when joining triple IRIs to document URIs.

Hidden columns in Optic are now returned if requested

The Optic engine has a number of hidden columns that it uses under the covers (e.g. `fragmentId`). Prior to MarkLogic 11, even if those columns were explicitly requested, they would not be returned. If those columns are requested in MarkLogic 11, they will now be returned.

5.7. Storage Failure Detection and Failover

MarkLogic 11 extends the existing high-availability features to detect when there are issues with storage, and optionally, trigger failover. MarkLogic 11 uses a background process to monitor all file

systems that store forest data. If there are significant delays accessing those file systems, MarkLogic will now log messages at progressively higher levels the longer the delays are.

- 5 to 10 seconds - Debug log message
  - `Debug: Forest::<forest name> read file <forest data directory>/DiskCheck hang for <x> seconds`
- >= 10 seconds - Warning log message
  - `Warning: Forest::<forest name> read file <forest data directory>/DiskCheck hang for <x> seconds`
- >= 60 seconds - Error log message
  - `Error: Forest::<forest name> read file <forest data directory>/DiskCheck hang for <x> seconds`
- >= 180 seconds - Critical log message
  - `Critical: Forest::<forest name> read file <forest data directory>/DiskCheck hang for <x> seconds`

Failover

Building on the new storage monitoring mechanism, MarkLogic 11 also provides the ability to trigger a failover if the storage cannot be accessed for a configurable amount of time. There are two new database configuration settings listed below that allow this to be controlled:

Setting	Description
shutdown-on-storage-failure	When set to true, if storage for any primary forest for the database cannot be accessed for more than the configured timeout, the MarkLogic process will shut itself down, triggering a failover of any forests that are being managed by the MarkLogic host. This is disabled by default. The following new admin functions can be used to enable or disable this behavior: • `admin.databaseGetShutdownOnStorageFailure()` • `admin.databaseSetShutdownOnStorageFailure()`
storage-failure-timeout	If "shutdown-on-storage-failure" is set to true, and storage cannot be accessed, this is the number of seconds to wait before triggering a MarkLogic shutdown. This is set to 60 seconds by default. The following new admin functions can be used to control this setting: • `admin.databaseGetStorageFailureTimeout()` • `admin.databaseSetStorageFailureTimeout()`

CAUTION

The "shutdown-on-storage-failure" setting should only be used for databases that have forests configured for high availability with local-disk failover. See Configuring Local-Disk Failover for a Forest in the *Scalability, Availability, and Failover Guide* for how to configure local-disk failover. **This feature should not be used with shared-disk failover.**

If a MarkLogic host has shut itself down due to a detected storage failure, procedures should be taken to repair the storage before restarting the MarkLogic process. Follow standard procedures for Reverting a Failed Over Forest Back to the Primary Host in the *Scalability, Availability, and Failover Guide*once the storage has been repaired and the MarkLogic host is restarted.

NOTICE
When MarkLogic shuts down due to a detected storage failure, it attempts to write a marker file to the default data directory (typically `/var/opt/MarkLogic`). When restarting, if MarkLogic sees this marker file, it will pause for two minutes prior to startup and then attempt to clear the marker file. This is to allow adequate time for failover to occur if there is an automated process in place to restart MarkLogic.

5.8. Enhanced Health Check Monitoring

NOTICE
There is a known issue when upgrading from MarkLogic 9 or 10 to 11.0.0. After upgrading, in order to enable the enhanced health check endpoints, edit the configuration of the "HealthCheck" app server and set the "Url Rewriter" to "rewriter.xml". This issue will be addressed in a coming patch release of MarkLogic Server.

In MarkLogic 11, the "health check" capability has been significantly enhanced. In versions prior to MarkLogic 11, there is a simple health check application server running on port 7997 that responds "Healthy" if the MarkLogic host is up and running. The primary use case for this existing service is for load balancer health probes.

In order to support higher-level monitoring of a host's and cluster's "health" and therefore, ability to service requests, MarkLogic 11 now provides health check at multiple levels:

API	Description
`http://<host>:7997`	Backward compatible and returns "Healthy" (200 OK) if the MarkLogic host is up and running and the HealthCheck application server is enabled. No authentication is required.
`http://<host>:7997/LATEST/healthcheck`	Returns "Healthy" (`200 OK`) only if the following conditions are met: • The target host has joined the cluster, and a quorum exists. • All the application servers are configured and ready to handle requests. • All the databases are mounted and available. • All server background processes are running without issue. Otherwise, returns "Unhealthy" (`503 Service Unavailable`). No authentication is required.

API	Description
`http://<host>:7997/LATEST/healthcheck/status`	Returns health status of each of the components. `<healthcheck-general-status xmlns="http://marklogic.com/xdmp/status/healthcheck-general"> <server-initializing>false</server-initializing> <server-restarting>false</server-restarting> <server-shutdown>false</server-shutdown> <quorum-exists>true</quorum-exists> <server-healthy>true</server-healthy> <background-healthy>true</background-healthy> <database-healthy>true</database-healthy> </healthcheck-general-status>` No authentication is required.
`http://<host>:8002/manage/v2?view=healthcheck`	Returns the detailed health status of each of the components checked. This API requires authentication as it exposes details of internal configuration. See /REST/GET/manage/v2@view=healthcheck for details.

Two new built-in functions (below) are available in MarkLogic 11 to support the new health check capability.

Function	Description
xdmp.clusterQuorumExists()	Returns whether a quorum exists in the specified cluster.
xdmp.healthcheckStatus()	Returns the health check status of each component in the current cluster.

5.9. HTTP Compression and Chunking

MarkLogic 11 adds out of the box support for automatic HTTP compression and chunking.

HTTP Compression

Prior to MarkLogic 11, HTTP compression on standard endpoints (such as /v1/rows) was not available. HTTP compression could be added to custom REST extensions via the use of xdmp.gzip to compress the response. However, this required custom code and lacked the ability to stream the response. For example, a large JSON document with an array with a million rows would have to be compressed in memory prior to returning the response.

MarkLogic 11 now supports compressing of data at the HTTP layer. This means that if clients specify the `Accept-Encoding: gzip` HTTP header, HTTP compression will be done for all out of the box endpoints. And, custom endpoints will not need to use custom code to compress data. Custom endpoints can leverage streaming results, and the HTTP layer will compress the results on the fly.

NOTICE

Custom endpoints that looked for the `Accept-Encoding: gzip` HTTP header and performed compression using custom code must remove that logic. If the logic is not removed, the results will be compressed twice, and not be usable by the client.

Custom HTTP endpoints can override the client request header and turn on or off the HTTP compression via a new built-in function xdmp.setResponseCompression(). This function also allows the compression level to be set.

Since most browsers send the `Accept-Encoding: gzip` HTTP header by default, all traffic to the MarkLogic built-in applications, such as the Admin UI and Query Console, will now be compressed.

HTTP Chunking

MarkLogic 11 adds the ability to use HTTP chunked transfer encoding. This adds the ability to report errors during the streaming of large results over HTTP.

HTTP chunking is activated in one of two ways:

1. The client includes the `TE:trailers` request header
2. Custom HTTP endpoints use the built-in function xdmp.setResponseChunkedEncoding(). Custom trailers can be added using xdmp.addResponseTrailer() function

If HTTP chunking has been activated by one of the above mechanisms, MarkLogic will send the following trailers (including the header):

- `ML-CONTENT-SHA256` - The SHA256 of the content that was sent (this can be used to check the all of the data that was sent was received)
- `ML-ERROR-CODE` - The error code if an error occurred, or "NONE" if there wasn't an error
- `ML-ERROR-MESSAGE` - The error message if an error occurred, or "NONE" if there wasn't an error
- Any custom trailers added via xdmp.addResponseTrailer()

5.10. UNNEST in Optic and SQL

MarkLogic 11 adds the ability to "unnest" rows from within rows in both SQL and the Optic API.

SQL

Support for the following SQL syntax is now available in MarkLogic 11:

```
CROSS|LEFT|INNER JOIN UNNEST(iterableExpression) [WITHORDINALITY]
[as<rename>]
```

`CROSS JOIN UNNEST` and `INNER JOIN UNNEST` behave the same and produce new rows for each of the nested values, but if the UNNEST input is NULL, no row will be output.

`LEFT JOIN UNNEST` produces new rows for each of the nested values, including a single row if the UNNEST input is NULL.

Optic API

The unnest capability is exposed in the Optic API via the two new Operators below:

Function	Description
op.unnestInner()	Flattens an array into multiple rows and performs an inner join against the rest of the rows
op.unnestLeftOuter()	Flattens an array into multiple rows and performs a left outer join against the rest of the rows

5.11. New CTS Document Queries

MarkLogic 11 adds three new CTS query constructors to make it easier to discover documents in a database:

Function	Description
cts.documentFormatQuery()	Matches documents with the given format. The format could be one of "json","xml","text", or "binary"
cts.documentRootQuery()	Matches document with the given root element QName
cts.documentPermissionQuery()	Matches documents that have been assigned the given permission

5.12. OAuth 2.0 External Authentication

MarkLogic 11 expands its external authentication options by adding support for OAuth 2.0. In this first phase of OAuth capability, MarkLogic 11 supports configuration of external security with OAuth servers

using “Introspection” access token validation. This capability has currently been validated against Ping Identity.

Support for other OAuth features, such as JWT access token validation, additional encryption options, and additional OAuth servers is planned for future releases.

OAuth external security can be setup via the **Security** -> **External Security** -> **Create** screen in the MarkLogic 11 Admin UI or via the sec.createExternalSecurity() and sec.oauthServer() built-in functions.

Also see Security-sec (sec.) for the complete list of built-in functions available.

5.13. Encryption Key for Database Backups

MarkLogic 11 now supports using the database encryption key for database backups. In versions prior to MarkLogic 11, encrypted database backups are encrypted using the cluster encryption key. In MarkLogic 11, this can be changed to use the database encryption key or the cluster encryption key. This setting is on a per-database basis, so each database can use its own key when encrypting backups. The default setting in MarkLogic 11 is to use the cluster encryption key.

See the following built-in functions for more details:

Function	Description
admin.databaseSetBackupEncryptionKey()	Sets whether the database key or cluster key will be used for encryption of backups for the given database. This can only be set when an external KMS is configured
admin.databaseGetBackupEncryption()	Gets whether the database key or cluster key will be used for encryption of backups for the given database

See Encryption at Rest in *Securing MarkLogic Server* for more details about the encryption key hierarchy used by MarkLogic.

5.14. XDQP Certificate Management

MarkLogic 11 adds a new set of APIs that allow administrators to better manage the certificates used for intra-cluster XDQP encryption (the group-level "xdqp ssl" settings) . The following new built-in functions can be used to check if hosts need new certs, to generate new certs, and then to activate the certs:

Function	Description
admin.hostNeedRenewXdqpCertificate()	Returns the list of hosts where the XDQP host certificate will expire within a specified duration
admin.hostRenewXdqpCertificate()	Generates a new XDQP host certificate for a host if the current certificate will expire within a specified duration
admin.hostActivateNewXdqpCertificate()	Activates the new XDQP host certificate that was previously generated by a call to admin.hostRenewXdqpCertificate()

5.15. LDAP Performance Improvement

MarkLogic 11 adds a negative cache capability for LDAP external security. This capability improves performance and reduces workload against LDAP services when multiple external security providers have been configured. The default negative cache timeout is 10 seconds, but this can be configured in the external security settings (see the external security REST API for specifics).

5.16. cts:estimate() in XQuery

The XQuery cts:estimate() API was added to be in parity with the JavaScript cts.estimate() API. This can be used in place of the longer `xdmp:estimate(cts:search())` syntax.

5.17. Install on macOS Running on Apple M1 Processors

MarkLogic 11 installation works on macOS running on Apple M1 processors. MarkLogic does not run natively on ARM, but it works well running under Rosetta 2 emulation. Prior to MarkLogic 11, the install scripts would error out indicating MarkLogic could only be installed on macOS running on an Intel chipset.

NOTE

MarkLogic is supported for development purposes only on macOS. See the Supported Platforms in *Installing MarkLogic Server* for details.

5.18. Updated S3 Interface Layer

In MarkLogic 11, a new S3 interface layer with added stability and compatibility is now the default for all required interactions with Amazon S3.

6. Known Incompatibilities with Previous Releases

Most applications implemented on previous versions of MarkLogic will run either without modifications or with very minor modifications in MarkLogic 11. This section summarizes product changes that can cause compatibility issues for applications developed using previous releases.

6.1. Version Number Format has Changed

As mentioned in the introduction of the release notes, beginning with version 11.0.0, MarkLogic Server is adopting the industry-standard version numbering system called Semantic Versioning. From this point forward, all public releases will use a three-part version string. The three parts represent, respectively, MAJOR, MINOR, and PATCH releases, with respect to compatibility of public APIs.

Because of this, the return string from xdmp.version() will look different. If you depend on the specific formatting of version numbers returned by that function, please adjust your scripts. In versions from 11.0.0 onward, it will return a string shaped like "11.0.0".

In versions prior to MarkLogic 11, xdmp.version() will continue to return a string shaped like "10.0-9.5".

If your application compares version numbers, be sure to follow the guidelines given at semver.org.

6.2. Deduplication is off by default for op.fromSPARQL/op:from-sparql

In MarkLogic 11, op.fromSPARQL() 's default deduplication option is "dedup=off". In versions prior to MarkLogic 11, the default was "dedup=on". This was changed to provide optimal performance and because most use cases can tolerate duplication at the triple level. Code that previously relied on deduplication of triples when calling op.fromSPARQL() will need to change to explicitly set the "dedup=on" option in MarkLogic 11.

See De-Duplication of SPARQL Results for more details about deduplication of triples in SPARQL.

6.3. JSON with Non-default Language Now Reindexed Properly

MarkLogic 11 contains a fix for a bug that caused JSON documents with non-default language properties to be reindexed incorrectly.

In versions prior to MarkLogic 11, incorrect search results may be returned after a reindex of JSON documents with the "lang" or "language" property set to a language other than the database's default language setting and stemming is enabled for the database.

For example, the following document, when inserted in a database with stemming enabled, would be indexed in the specified language--in this case, "fr" (French) even if the default database language is not set to the document's language:

```
{
 "lang": "fr",
 "type": "bonjour"
}
```

A query with a specified matching language--in this case "fr"--will match this document:

```
cts:search(fn:doc(),cts:json-property-value-query("type", "bonjour","lang=fr"))
```

However, upon reindexing, if the default database language is not set to "fr", documents with a language property other than the default language, as in the example, will no longer match a query with a non-default language. For example, after reindexing occurs, the following query would stop matching the example document:

```
cts:search(fn:doc(),cts:json-property-value-query("type", "bonjour","lang=fr"))
```

However, this query, which used the default database language, would incorrectly match the example document:

```
cts:search(fn:doc(),cts:json-property-value-query("type", "bonjour"))
```

After upgrading to MarkLogic 11, if there are JSON documents with non-default language properties in databases with stemming enabled, you will not notice a change in behavior unless a reindex is triggered either manually or by a change in the index configuration. If a reindex is performed, queries that were not working correctly before, i.e. queries that should have been matching documents with non-default language properties, will now work correctly, and there may be additional matching documents.

6.4. HTTP Compression is on by default

As mentioned in HTTP Compression and Chunking, MarkLogic 11 now supports HTTP compression for all HTTP endpoints when clients include the `Accept-Encoding: gzip` HTTP header. Custom HTTP endpoints written for versions prior to MarkLogic 11 had the ability to look for the `Accept-Encoding: gzip` HTTP header and perform compression of the results themselves using built-in functions. When running these custom endpoints in MarkLogic 11, you must remove the logic in the custom code that does compression code or the results will be compressed twice and not be usable by clients.

6.5. op.bindAs/op:bind-as No Longer Supported

The `op.bindAs()/op:bind-as()` Optic function was deprecated in an earlier release of MarkLogic 10 in favor of op.bind() combined with op.as(). The `op.bindAs()/op:bind-as()` function is not supported in MarkLogic 11. There is a known issue that the documentation for the function is still available. This will be addressed in a future documentation update.

6.6. Objects are Now Exported Deeply in Optic

In versions prior to MarkLogic 11, a key-value object was exported as itself. In MarkLogic 11, the keys and values will be examined and exported one by one. Nested objects will be exported as well.

For example, this query

```
op.fromLexicons({
             uri:cts.uriReference(),
             number:cts.jsonPropertyReference('srchNumber'),
             city:cts.jsonPropertyReference('srchCity')
           },
           'docRange')
       .export();
```

would have been exported like this in versions prior to MarkLogic 11:

```
{$optic:{ns:'op', fn:'operators', args:[
    {ns:'op', fn:'from-lexicons', args:[{
             uri:    {uriReference:{}},
             number: {jsonPropertyReference:{property:'srchNumber', scalarType:'int',
nullable:false}},
             city:   {jsonPropertyReference:{property:'srchCity',   scalarType:'string',
nullable:false,
                      collation:'http://marklogic.com/collation/'}}
       },
       'docRange']}
     ]}};
```

In MarkLogic 11, it is exported like this:

```
{$optic:{ns:'op', fn:'operators', args:[
{ns:'op', fn:'from-lexicons', args:[{
        uri:    {ns: 'cts', fn: 'uri-reference', args: []},
        number: {ns: "cts",
          fn: "json-property-reference",
          args: ["srchNumber", ["type=int"]]
        },
        city:   {ns: "cts",
                 fn: "json-property-reference",
                 args: ["srchCity",
                         ["type=string", "collation=http://marklogic.com/collation/"]
                 ]
               }
        },
        'docRange']}
  ]}};
```

6.7. Entity Services Features Deprecation Extended

The XML representation of an entity type and some Entity Services functions were deprecated in 10.0-4 and 10.0-5 with the intention to drop support in 11. However, the deprecation period has been extended to MarkLogic 12.

6.8. HDFS Removed

HDFS support was deprecated in MarkLogic 10.0-3 and removed in 11.0.0.

6.9. Ops Director Removed

The Ops Director feature was deprecated in MarkLogic 10.0-5 and is now removed as of MarkLogic 11.

6.10. MarkLogic 10 Incompatibilities

6.10.1. Incompatibilities between 10.0-5 and 10.0-4

Reindexing After 10.0-5 Upgrade

A fix made in 10.0-5 may cause reindexing after the server is upgraded to 10.0-5. If you have a server configuration where a named field in a database is configured for value searches and stemming is turned *off*, while the database itself has stemming is turned *on*, reindexing will occur after the upgrade to 10.0-5 is completed.

Ops Director Deprecated

The Ops Director feature is deprecated in MarkLogic 10.0-5.

Hadoop Connector Removed

The Hadoop Connector was deprecated starting with MarkLogic 10.0-4 and is removed from the product in MarkLogic 10.0-5.

Change to Calling SUM or AVG on String-Typed Column

As of MarkLogic Server 10.0-5, in SQL, SPARQL, and Optic, `XDMP-EXPR` is thrown when calling `SUM()` or `AVG()` on a string-typed column argument.

Entity Services Features Deprecated

The XML representation of an entity type has been deprecated in 10.0-5.

The following Entity Services functions have been deprecated in 10.0-5 because DHF provides the equivalent functionality. These functions will not be supported in MarkLogic 11.

JavaScript	XQuery
es.databasePropertiesGenerate	es:database-properties-generate
es.extractionTemplateGenerate	es:extraction-template-generate
	es:optional
es.piiGenerate	es:pii-generate
es.schemaGenerate	es:schema-generate
es.searchOptionsGenerate	es:search-options-generate

6.10.2. Incompatibilities between 10.0-4 and 10.0-3

mlcp Distributed Mode Deprecated

mlcp distributed mode has been deprecated for 10.0-4.

HDFS Support Deprecated

HDFS support has been deprecated in MarkLogic 10.0-3.

Priority to Determine MarkLogic License Key

MarkLogic prioritizes the license key value from the following locations in the following order. The version number determines which locations take precedence.

NOTE
The `mlcmd.conf` location is only considered when MarkLogic is running on Amazon Web Services (AWS).

10.0-4:

Admin Interface (`server.xml`)

1. `/etc/marklogic.conf`
2. `mlcmd.conf` (AWS CloudFormation template user data)

10.0-3:

`mlcmd.conf` (AWS CloudFormation template user data)

1. `/etc/marklogic.conf`
2. Admin Interface (`server.xml`)

Entity Services Features Deprecated

The following Entity Services functions were deprecated in 10.0-4 because DHF provides the equivalent functionality. These functions will not be supported in MarkLogic 11.

JavaScript	XQuery
es.instanceConverterGenerate	es:instance-converter-generate
es.instanceFromDocument	es:instance-from-document
es.instanceGetAttachments	es:instance-get-attachments
es.instanceJsonFromDocument	es:instance-json-from-document

JavaScript	XQuery
es.instanceXmlFromDocument	es:instance-xml-from-document
es.modelFromXml	es:model-from-xml
es.modelGetTestInstances	es:model-get-test-instances
es.modelToXml	es:model-to-xml
	es:add-attachments
	es:copy-attachments
	es:serialize-attachments
	es:init-instance
	es:init-source
	es:init-translation-source
es.versionTranslatorGenerate	es:version-translator-generate
	es:extract-array
	es:with-namespace

6.10.3. Incompatibilities between 10.0-3 and 10.0-2

Privilege escalation allowing execution of xdmp:data-directory()

Certain privileged built-in functions, for example `xdmp:data-directory` and `xdmp:list-cache-size`, that return static environmental data may be prematurely optimized out with the results in-lined as literals in code passed to `xdmp:invoke-function` or `xdmp:spawn-function`.

The in-lining occurs in the outer environment not the inner environment, so the execution privileges checked are of the outer environment not the inner environment. As a result, even if the inner environment does not have privileges to execute the built-in functions, no exception is thrown when the optimized code is run. This issue is addressed in 10.0-3.

The API trgr:triggers-change-modules-database() Only Has Two Parameters

The API `trgr:triggers-change-modules-database()` only takes two parameters in 10.0-3.

1. the old database as `xs:anyAtomicType`. This can be an ID `unsignedLong` or a name `xs:string`

2. the new database as `xs:anyAtomicType`. This can be a database's ID `unsignedLong` or a database's name `xs:string`.

For more details and sample code, see our API documentation.

Configuration Manager Removed

The Configuration Manager tool is deprecated starting with MarkLogic 9.0-5 and removed from MarkLogic Server in version 10.0-3.

6.10.4. Incompatibilities between 10.0-2 and 10.0-1

Module and Library Removed

Beginning with version 10.0-2, the `mustache.xqy` module and library existing under the folder `MarkLogic/Modules/MarkLogic/mustache/` is no longer available.

Encryption Change

Data encrypted by the embedded KMS in MarkLogic 10.0-2 cannot be decrypted using versions prior to MarkLogic 10.0-2. Exporting the internal wallet requires a passphrase that is now encrypted by argon2, where previously the files were encrypted with crypt. Additionally, backups encrypted with only a passphrase in MarkLogic 10.0-2 cannot be restored in versions prior to 10.0-2 for the same reason.

6.10.5. Incompatibilities Between 10.0-1 and MarkLogic 9

Packaging API Removed

In MarkLogic 9, the Packaging API was deprecated. It has been removed from the product in MarkLogic 10.

For information about the deprecated endpoints and the alternative endpoints to use instead, see

The following table lists the deprecated endpoints and the new alternative.

Deprecated Endpoints	Alternative
• `GET /manage/v2/packages/{pkgname}` • `POST /manage/v2/packages` • `GET /manage/v2/packages` • `POST /manage/v2/packages/{pkgname}` • `HEAD /manage/v2/packages/{pkgname}` • `DELETE /manage/v2/packages/{pkgname}` • `GET /manage/v2/packages/{pkgname}/databases` • `GET /manage/v2/packages/{pkgname}/databases/{name}` • `POST /manage/v2/packages/{pkgname}/databases/{name}` • `HEAD /manage/v2/packages/{pkgname}/databases/{name}` • `DELETE /manage/v2/packages/{pkgname}/databases/{name}` • `POST /manage/v2/packages/{pkgname}/install` • `GET /manage/v2/packages/{pkgname}/servers` • `GET /manage/v2/packages/{pkgname}/servers/{name}` • `POST /manage/v2/packages/{pkgname}/servers/{name}` • `HEAD /manage/v2/packages/{pkgname}/servers/{name}` • `DELETE /manage/v2/packages/{pkgname}/servers/{name}`	Use the following endpoints of the CMA REST API instead: • `GET /manage/v3` • `POST /manage/v3`
Base Path: `/manage/v2/tickets:` • `POST /manage/v2/tickets/{ticketnumber}/revert`	

Change to Admin Role

In MarkLogic 9, a user with the `admin` role was automatically able to see template driven extraction (TDE) views that were read-protected. In MarkLogic 10, a user with the `admin` role cannot see TDE views unless explicitly granted read access.

XQuery 0.9-ml Deprecated

The XQuery `0.9-ml` dialect has been deprecated and will eventually no longer be supported. MarkLogic recommends that you port any legacy `0.9-ml` XQuery code to either enhanced XQuery dialect (`1.0-ml`) or strict XQuery dialect (`1.0`). For more information, see Porting 0.9-ml XQuery Code to Enhanced 1.0-ml in the *XQuery and XSLT Reference Guide*.

Template Driven Extraction (TDE) functions are also affected by this deprecation. The function `cts:valid-index-path` may throw error messages if you do not upgrade your code to XQuery dialect `1.0-ml` or `1.0`.

JavaScript: Calling the seal() Function on MarkLogic Wrapped Objects Now Raises an Error

The following code will now throw an error because fn.doc("/doc-1.json").toObject()[0].root.d is a MarkLogic wrapped object.

```
var r = fn.doc("/doc-1.json").toObject()[0].root.d
  r.beforeSeal = "hello"
    var before =  r.beforeSeal;
      Object.seal(r)
=>
Error running JavaScript request: TypeError: Cannot seal
```

Change in Default rest-reader and rest-writer Permissions

To enable the creation of a document by a user who does not have `rest-reader` or `rest-writer` privileges, the following backward-incompatible changes were made to the REST API to improve security:

Prior to MarkLogic 10.0-1, when inserting documents, the REST API assigned permissions based on the default permissions configured for the user and role but also assigned read permissions to the `rest-reader` role and assigned update permissions to the `rest-writer` role. As a result, any user with the `rest-reader` role had permission to read to read all documents and any user with the `rest-writer` role had permission to update all documents.

In MarkLogic 10.0-1, when inserting documents, the REST API assigns permissions based only on the default permissions configured for the user and role. As a result, it is possible to adopt a security model in the REST API where no role has access to all documents.

In other words MarkLogic 9, when you wrote documents with the REST API using PUT v1/documents, the documents had the union of the following permissions:

- Any permissions specified in the request
- The default permissions for the user and document or, when update policy is set to overwrite-metadata, the existing permissions on the document
- Read permission for the rest-reader role and update permissions for the rest-writer role

In MarkLogic 10, when you write documents with the REST API using PUT v1/documents, the documents have the union of the following permissions:

- Any permissions specified in the request
- The default permissions for the user and document or, when update policy is set to overwrite-metadata, the existing permissions on the document

What has changed is that the `rest-reader` and `rest-writer` convenience roles no longer have any permissions on a document unless one of the following is true:

- The request specifies permissions for the `rest-reader` or `rest-writer` role.
- The definition of the user grants default permissions to the `rest-reader` or `rest-writer`.

Since the `rest-writer` convenience role default permissions grant reader permission to the `rest-reader` role and update permissions to the `rest-writer` role, documents written by a user who has the `rest-writer` convenience role are readable by users with the rest-reader role and writable by users with the `rest-writer` role.

- In MarkLogic 9, a user given the `rest-reader` or `rest-writer` role had access to every document written with the REST API.
- In MarkLogic 10, a security model need not grant any role access to every document written with the REST API. Documents inserted by users with the `rest-writer` role still have read permissions for the `rest-reader` role and update permissions for the `rest-writer` role.

To override this backward incompatibility, you must modify the user role creating documents to give default permissions to `rest-writer` and `rest-reader`.

Certificate Authority Handling Changed During Upgrade

When you upgrade to MarkLogic Server 10, the CAs defined by External Securities are merged into their appropriate App Server's CA list. This consolidates the functionality of where an App Server admin defines CAs to accept as the signer of client certificates.

Changed Functions

The following functions have been changed as of MarkLogic 10:

- sec:create-external-security
  This function no longer takes the arguments `client-certificate-authorities` and `require-client-certificate`.

For more information, see External Certificate User Authentication in *Securing MarkLogic Server*.

Deprecated Functions

The following functions have been deprecated as of MarkLogic 10 and will be removed in a future release:

- `external-security-set-ssl-client-certificate-authorities`
- `external-security-set-ssl-require-client-certificate`
- `external-security-get-ssl-client-certificate-authorities`
- `external-security-get-ssl-require-client-certificate`

Please use the following functions instead:

- `appserver-set-ssl-client-certificate-authorities`
- `appserver-set-ssl-require-client-certificate`
- `appserver-get-ssl-client-certificate-authorities`
- `appserver-get-ssl-require-client-certificate`

6.11. MarkLogic 9 Incompatibilities

This section describes the incompatibilities between MarkLogic 8 and MarkLogic 9. This is here just for convenience. For the full MarkLogic 9 *Release Notes*, see docs.marklogic.com/9.0/guide/relnotes.

6.11.1. JavaScript: ValueIterator Replaced By Sequence

The `ValueIterator` interface used to represent sequences of value in MarkLogic 8 has been replaced by the new `Sequence` interface. A `Sequence` is a JavaScript Iterable object. All functions which previously operated on or returned a `ValueIterator` now use a `Sequence` instead.

In many cases, this change is transparent to your code. However, code that depends on the following `ValueIterator` properties and methods must be changed:

- `ValueIterator.next` - Use a `for..of` loop to iterate over a `Sequence`. Use `fn.head` if you just want to pick off the first or only value in a Sequence.
- `ValueIterator.count` - Use `fn.count` instead.
- `ValueIterator.clone` - No longer needed. You can iterate over the same `Sequence` multiple times.

For more details, see Sequence in the *JavaScript Reference Guide* and Sequence in the *MarkLogic Server-Side JavaScript Function Reference*.

6.11.2. Database Stemming is Off, Word Searches On By Default

In MarkLogic 9 and later, when you create a new database, the `stemmed searches` property is `off` by default. In MarkLogic 8 and earlier, the default is `basic`.

In MarkLogic 9 and later, when you create a new database, `word searches` are enabled by default. In MarkLogic 8 and earlier releases, `word searches` were disabled by default.

To achieve the pre-MarkLogic 9 default behavior, configure your database to turn off `stemmed searches` and set `word searches` to `true`.

These changes only affect databases you create after upgrading to MarkLogic 9. Databases that exist when you upgrade will retain their previous settings.

6.11.3. Collection Lexicon and Triple Index Enabled by Default

In MarkLogic 9, a fresh install of MarkLogic 9 will enable the triple index and collection lexicon for all databases. Databases that exist when you upgrade to MarkLogic 9 will retain their previous settings. Any new databases created after upgrading to MarkLogic 9 will have the triple index and collection lexicon enabled.

6.11.4. XCC .NET API No Longer Available

The XCC .NET interfaces have been removed from MarkLogic. Current users of XCC .NET are encouraged to use the REST Client API to create equivalent interfaces. For details, see the REST Application Developer's Guide.

The XCC Java Library continues to be available.

6.11.5. Changes in Semantic Query Behavior

MarkLogic 9 introduced changes to the behavior of semantic queries.

Triple Index and SPARQL Engine Changes

The triple index and SPARQL engine were changed in MarkLogic 9. Now as part of a query, two triples are considered equal if their subjects, predicates, and objects compare as equal with the SPARQL "=" operator.

Previously, these triples would be treated as two different, non-identical triples:

```
sem:triple(xs:anyURI("http://a"), xs:anyURI("http://b"), xs:anyURI("http://c")),
sem:triple("http://a", "http://b", "http://c").
```

In MarkLogic 9 and later, values of type `xs:string` and `xs:anyURI` can compare equal if they have the same lexical form according to the SPARQL "=" operator. This functionality is called D-entailment (D as in datatype). The two triples in the example are now considered to be the same, and the de-duplication process removes the duplicate. If you return frequencies, you would see the second triple show up as one extra in the frequencies.

Also note that the query may return either `sem:triple(xs:anyURI("http://a"), xs:anyURI("http://b"), xs:anyURI("http://c"))`, or `sem:triple("http://a", "http://b", "http://c")` as the result since they are considered equal.

Forest IDs Removed From sem:sparql Function

The forest ID options for the `sem:sparql` function, which were part of MarkLogic 7, were removed in MarkLogic 8, but their functionality was kept for backwards compatibility. In MarkLogic 9, the forest ID option will no longer work with `sem:sparql`.

6.11.6. Triple Count Increased After Inserting Same Data Twice

During a rolling upgrade, while the cluster is in a mixed mode (not all hosts committed to the new version yet) the triple count of semantic triples may be increased after inserting the same data twice. During a rolling upgrade, the MarkLogic 9 and later triple index is not able to return triples in the correct order for MarkLogic 8 semantics. For this situation to occur, a user would need to have multiple triples that are identical except for the types of the values.

6.11.7. Database max merge size Now Defaults to 48 GB

The database parameter `max merge size` defaults to 48 GB (49152 MB) for new databases created in MarkLogic 9 and later. Previously, the default was 32 GB. Existing databases will keep whatever `max merge size` setting is configured after upgrading to MarkLogic 9. The new setting reflects advances in storage systems and should be appropriate for most databases.

6.11.8. Changes to Range Index Reference Resolution

In MarkLogic 9 and later, index reference resolution in range index construction, query constructors, and lexicon operations succeed in some cases that would have thrown an `XDMP-RIDXNOTFOUND` error

in previous releases. If the index reference contains enough information to unambiguously match an existing index, MarkLogic will do so. Previously, MarkLogic assumed default values for some “missing” index attributes, resulting in an error.

Code that previously succeeded will continue to do so. Some code that would previously have gotten an error will no longer do so.

For example, suppose you define a geospatial element range index on element `xs:QName("coords")` with type `long-lat-point` and coordinate system `wgs84`, and then refer to the index by just the element QName (`cts:element-geospatial-values(xs:QName("coord"))`). Previously, the reference would have been an error because the default point type (`point`) would have been assumed. Now, the reference resolves correctly as long as there is not a second geospatial element range index on the same QName with different coordinate system or point type.

When an ambiguous range index reference is detected, MarkLogic 9 throws one of the new exceptions `XDMP-RIDXAMBIGUOUS` (range index) or `XDMP-GIDXAMBIGUOUS` (geospatial index).

6.11.9. Default Stemming and Tokenization Libraries Changed for Most Languages

In MarkLogic 9 and later, the default tokenization and stemming libraries have been changed for all languages (except English) tokenization. Consequently, some tokenization and stemming behavior changed between MarkLogic 8 and MarkLogic 9. In most cases, stemming and tokenization will be more precise in MarkLogic 9 and later.

If you upgrade to MarkLogic 9 or later from an earlier version of MarkLogic, your installation will continue to use the legacy stemming and tokenization libraries as the language baseline. Any fresh installation of MarkLogic will use the new libraries. You can change the baseline configuration using `admin:cluster-set-language-baseline`.

NOTE

- Changing the baseline requires a cluster-wide restart and a reindex to avoid stemming and tokenization anomalies.
- Use of the legacy libraries is deprecated. These libraries will be removed from MarkLogic in a future release.

Unless you use the legacy language baseline, reindexing is required for content in the following languages:

- Chinese
- Danish
- Dutch, if you want to query with decompounding
- Finnish
- German
- Hungarian
- Japanese
- Korean, unless you use decompounding
- Norwegian (Bokmal and Nynorsk) if you want to query with decompounding
- Norwegian (generic ‘no’ lang code), though use of generic ‘no’ is not recommended
- Romanian

- Russian
- Swedish, if you want to query with decompounding
- Tamil
- Turkish

For other languages (except English), you might be able to avoid incompatibilities depending on the nature of your queries, but reindexing is still strongly recommended.

Tokenization and stemming are significantly different for Japanese. Tokenization is significantly different for Chinese (both simplified and traditional). The impact on other languages is more nuanced, but should lead to better results, overall. You might observe some relevance score changes on stemmed searches due to the higher degree of precision. If you require low-level details about the impact on a specific language and you have an active maintenance contract, you can contact MarkLogic Technical Support.

For more details on incompatibilities related to the changes to stemming and tokenization, see MarkLogic Server v9 Tokenization and Stemming from MarkLogic Technical Support.

For more information about the new tokenization and stemming support, see Section 6.11.9, “Default Stemming and Tokenization Libraries Changed for Most Languages” [40].

6.11.10. SQL DESCRIBE No Longer Supported by xdmp:sql

In MarkLogic 9, the SQL `DESCRIBE` function was no longer supported. MarkLogic does support `DESCRIBE` queries over ODBC, but not from `xdmp:sql()`.

6.11.11. Application-Specific Logging

MarkLogic 9 and later provide application-specific access to logging. Instead of logging all errors to `ErrorLog.txt`, the `xdmp:log` function now writes to an app-server-specific log file - for example `8000_ErrorLog.txt` or `8020_ErrorLog.txt`. Anything event running on the task server is logged to `TaskServer_ErrorLog.txt`. The `ErrorLog.txt` file is now for MarkLogic “system” logging only.

Splitting the log files in this manner enables customer log files (which could potentially contain confidential information) to be separated from system log files. This helps ensure the privacy of customer data when they interact with Support and use the new Telemetry feature.

In a UNIX environment you can watch the multiple log files using something like this:

```
tail -f /path/file1 /path/file2 etc.
```

6.11.12. Change to Classification of Some Special Symbol Tokens

The classification of some symbols changed for purposes of tokenization as of MarkLogic 9, including the symbols in the following table. These changes can affect search results.

Symbols	Old Classification	New Classification
spacing accents(5E, 60, A8, AF, B4, B8)	punctuation	diacritic
copyright (A9)registered (AE)degree (B0)	punctuation	symbol
Spanish masculine/feminine ordinals(AA, BA)	punctuation	diacritic
superscript numbers(B2, B3, B9)	punctuation	diacritic
micro (B5)	punctuation	greek
fractions (BC, BD, BE)	punctuation	symbol

6.11.13. Change to xdmp:user-last-login

The `userid` parameter has been removed from `xdmp:user-last-login`. In MarkLogic 8 the `xdmp:user-last-login` took one parameter (`userid`). If the `userid` was different from the current user, the function returned an empty sequence. If the `userid` was the same as that of the current user, it only returned the last login information.

In MarkLogic 9 the function does not take a parameter. The `xdmp:user-last-login` function only returns the last login information for the current user.

6.11.14. Changed Interfaces for xdmp:document-insert and xdmp:document-load

These two functions, `xdmp:document-insert` and `xdmp:document-load`, significantly changed interfaces in MarkLogic 9. The interfaces will still work for purposes of backward compatibility, but are no longer documented.

6.11.15. search:parse Returns a Different Type for cts:query Output Format

In MarkLogic 8, `search:parse` returned the XML serialization of a cts:query by default. You could also explicitly select this return type by specifying “cts:query” as the value of the `$output` parameter.

In MarkLogic 9 and later, if you explicitly specify “cts:query” as the value of the `$output` parameter, you will now get a cts:query object instead of a serialized query. The default return type from `search:parse` is unchanged.

If your code explicitly sets the output type to “cts:query” and passes the output of `search:parse` straight through to functions such as `search:resolve` or `cts:search`, no code changes are required.

If your code explicitly sets the output type to “cts:query” and then transforms or traverses the output query XML, then you must change your code to use the new “schema-element(cts:query)” for the `$output` parameter of `search:parse`.

For example, the following call generates the XML serialization of a cts:query in MarkLogic 9 and later:

```
search:parse("cat AND dog", (), "schema-element(cts:query)")
```

6.11.16. Default Client API Search Behavior Change on Port 8000

The defined default search behavior for the Java, Node.js, and REST Client APIs is unfiltered search unless you override the default with your own options. Prior to MarkLogic 9, this default behavior was not honored when you used the Client APIs to interact with MarkLogic through the pre-defined App Server on port 8000.

As of MarkLogic 9, searches via the Client APIs on port 8000 default to unfiltered search. To get the old behavior, use query options that explicitly specify filtered search.

6.11.17. JSON Property Scope and Container Queries Match Array Items Differently

In MarkLogic 8, the query constructed by the XQuery function `cts:json-property-scope-query` and the Server-Side JavaScript function `cts.jsonPropertyScopeQuery` matched if its criteria query matched anywhere within the configured scope. In MarkLogic 9 and later, the behavior changed for certain JSON property scope queries where the scope property value is an array of objects.

The behavior has changed for searches that have the following characteristics:

- The `query` parameter of the property scope query is an and-query. Such a scope query effectively finds co-occurrences of matches to the and-query criteria within the specified scope.
- In the document to be matched, the value of the scope property is an array of objects. The item type is determined by examining only the first item in the array.

Given this setup, in MarkLogic 9 and later, the query only matches if all the sub-queries of the and-query occur in the same array item. In MarkLogic 8, the query matches even when the and-query matches occur in different array items. All other forms of JSON property scope query are unchanged.

This change makes it possible to construct a JSON property scope query that constrains matches to occurrences with a single array item. In MarkLogic 8, this was not possible.

This change also affects the `container-query` element of a structured query and QBE container queries.

The following example query is of the form affected by this change. It finds co-occurrences of “prop1” with value “value1” and “prop2” with “value2” within the scope of “root”. (The and-query criteria can be arbitrarily complex.)

Language	Example Query

XQuery	`cts:json-property-scope-query(`
	`  "root",`
	`  cts:and-query((`
	`  cts:json-property-value-query("prop1", "value1"),`
	`  cts:json-property-value-query("prop2", "value2")`
	`  ))`
	`)`

Server-Side JavaScript	`cts.jsonPropertyScopeQuery(`
	`  'root',`
	`  cts.andQuery([`
	`  cts.jsonPropertyValueQuery('prop1', 'value1'),`
	`  cts.jsonPropertyValueQuery('prop2', 'value2')`
	`  ])`
	`);`

If you search the following documents with the previous query shown, you get the results shown. The JSON properties that match the and-query criteria are shown in bold.

Sample Document	MarkLogic 8	MarkLogic 9

`// (1) criteria met in different array items`
`{"root": [`
`  { "prop1": "value1", "prop2": "v" },`
`  { "prop1": "v", "prop2": "value2" }]}`
	Match	No match

`// (2) criteria met in the same array item`
`{"root": [`
`  { "prop1": "value1", "prop2": "value2" },`
`  { "prop1": "v", "prop2": "v" }`
`]}`
	Match	Match

`// (3) criteria met in a child property`
`{"root": {`
`  "child": [`
`  { "prop1": "value1", "prop2": "v" },`
`  { "prop1": "v", "prop2": "value2" } ]`
`}}`
	Match	Match

Document (1) does not match in MarkLogic 9 and later because both and-query criteria are not satisfied in the same array item. Document (2) matches in MarkLogic 9 and later because both and-query criteria are satisfied in the same array item. The Document (3) results are unaffected because the value of the “root” property is not an array of objects.

You can restore the MarkLogic 8 behavior by using an and-query of multiple JSON property scope queries. For example, the following query matches Document (1) in MarkLogic 9 and later:

Language	Example Query

XQuery	`cts:and-query((`
	`  cts:json-property-scope-query(`
	`  "root", cts:json-property-value-query("prop1", "value1")),`
	`  cts:json-property-scope-query(`
	`  "root", cts:json-property-value-query("prop2", "value2"))`
	`))`

Server-Side JavaScript	`cts.andQuery([`
	`  cts.jsonPropertyScopeQuery(`
	`  'root', cts.jsonPropertyValueQuery('prop1', 'value1')),`
	`  cts.jsonPropertyScopeQuery(`
	`  'root', cts.jsonPropertyValueQuery('prop2', 'value2'))`
	`]);`

To ensure the modified queries match only when the and-query matches occur in the same instance of the scope property (“root” in the above examples), wrap it in JSON property scope query on the parent property. For example:

Language	Example Query

XQuery	`cts:json-property-scope-query("parent-of-root",`
	`  cts:and-query((`
	`  cts:json-property-scope-query(`
	`  "root", cts:json-property-value-query("prop1", "value1")),`
	`  cts:json-property-scope-query(`
	`  "root", cts:json-property-value-query("prop2", "value2"))`
	`)))`

Server-Side JavaScript	`cts.jsonPropertyScopeQuery('parentOfRoot',`
	`  cts.andQuery([`
	`  cts.jsonPropertyScopeQuery(`
	`  'root', cts.jsonPropertyValueQuery('prop1', 'value1')),`
	`  cts.jsonPropertyScopeQuery(`
	`  'root', cts.jsonPropertyValueQuery('prop2', 'value2'))`
	`]));`

6.11.18. REST Client API Incompatibilities

MarkLogic 9 introduced some backward-incompatible changes to the REST Client API.

keyvalue Service Removed

The previously deprecated `/keyvalue` is no longer available. That is, the method GET `/v1/keyvalue` is no longer available.

To perform similar operations, use the `/search` service with a structured query or combined query, or the `/qbe` service.

Collections in Request Parameters are OR Related

When you specify multiple collections through the collection request parameter of the following methods, they are now OR related:

- GET and POST /v1/search
- GET and POST /v1/qbe
- GET and POST /v1/values/*name*

The new behavior is consistent with the semantics of `cts:collection-query` and the structured query `collection-query`.

To get AND semantics, construct your own `and-query` of collection URIs as part of a structured or combined query, or by defining a constraint in your query options.

The new behavior differs from previous behavior in the following way:

- In MarkLogic 8.0-6, the GET methods applied AND semantics to multiple collection parameters, while the POST methods applied OR semantics.
- In MarkLogic 8.0-5 and earlier, both GET and POST methods applied AND semantics to multiple collection parameters.

Default value of Document Management “repair” parameter changed

The default value of the `repair` parameter in the Document Management PUT /v1/documents function has changed in the following way:

- In Marklogic 8.0 and earlier, the default value for `repair` is `full`.
- In Marklogic 9.0 and later, the default value for `repair` is `none`.

6.11.19. Java Client API Incompatibilities

MarkLogic 9 introduced some backward-incompatible changes to the Java Client API.

Java Client API: Removal of Deprecated Interfaces

Most of the previously deprecated packages, interfaces, classes, and methods of the Java Client API have been removed. The following table lists what has been removed and provides guidance for modifying your code.

NOTE

In the following table, “c.m.c.” is an abbreviation for “com.marklogic.client.”.

Removed Package, Class, or Method	Alternative
`c.m.c.admin.ServerConfigurationManager:` • `getContentVersionRequestsset` • `ContentVersionRequests`	Use the following equivalent methods instead. • `getUpdatePolicy` • `setUpdatePolicy`
`c.m.c.admin.TransformExtensionsManager:` • `writeXqueryTransform` • `writeXSLTransform`	Overloads of the listed methods that accept a `paramTypes` map have been removed. Use one of the overloads that does not accept parameter metadata. This metadata is not required to install or use a transform.
`c.m.c.admin.config`	This package, its sub-packages, and all contained classes, interfaces, and types have been removed. This includes the `QueryOptions` and `QueryOptionsBuilder` classes. Create query options using your preferred JSON or XML libraries, read options from a file, or build options as a string. Read and write options using appropriate handles, such as `DOMHandle`, `JacksonHandle`, `FileHandle`, or `StringHandle`.
`c.m.c.document.JSONDocumentManager:` • `getLanguage` • `setLanguage`	Remove your usage. Language specifications are not supported for JSON. Calls to these methods were ignored in MarkLogic 8.
`c.m.c.io.QueryOptionsHandle`	Create query options using your preferred JSON or XML libraries, read options from a file, or build options as a string. Read and write options using appropriate handles, such as `DOMHandle`, `JacksonHandle`, `FileHandle`, or `StringHandle`. This class is no longer needed with the removal of QueryOptions and QueryOptionsBuilder.
c.m.c.query: • `KeyValueDefinition` • `KeyLocator` • `ElementLocator`	The key-value search capability has been removed. Build equivalent queries using a `StructuredQueryDefinition` (JSON property query or element query) or QBE.
c.m.c.query.QueryManager: • `newKeyValueDefinition` • `newKeyValueDefinition` • `newKeyLocator`	The key-value search capability has been removed. Build equivalent queries using a StructuredQueryDefinition (JSON property query or element query) or QBE. c.
`c.m.c.io.SearchHandle.forceDOM`	Remove your usage. This setting was ignored in MarkLogic 8.

`c.m.c.query.StructuredQueryBuilder:`
 	
`FragmentScope.DOCUMENT`
	Use `FragmentScope.DOCUMENTS`.

Removed Package, Class, or Method	Alternative
StructuredQueryBuilder (nested classes): • `AndNotQuery` • `AndQuery` • `BoostQuery` • `CollectionConstraintQuery` • `CollectionQuery` • `CustomConstraintQuery` • `DirectoryQuery` • `DocumentFragmentQuery` • `DocumentQuery` • `ElementConstraintQuery` • `GeospatialConstraintQuery` • `LocksQuery` • `NearQuery` • `NotQuery` • `OrQuery` • `OrQuery` • `PropertiesConstraintQuery` • `PropertiesQuery` • `RangeConstraintQuery` • `TermQuery` • `ValueConstraintQuery` • `WordConstraintQuery`	Continue to construct structured queries using StructuredQueryDefinition methods, as before. The return type from the query builder methods is always StructuredQueryDefinition now. For example, `StructuredQueryDefinition.and()` used to return an `AndQuery`, but now returns a `StructuredQueryDefinition`.
`StructuredQueryBuilde.elementConstraint`	`StructuredQueryBuilder.containerConstraint`

Java Client API: JAR File Name and Maven Artifact ID Change

The JAR file for the Java Client API is now named `marklogic-client-api-`*version*`.jar`. Previously, the name was `java-client-api-`*version*`.jar`. For example, the JAR file name for version 3.0.6 is `java-client-api-3.0.6.jar`, but the JAR file name for version 4.0.1 is `marklogic-client-api-4.0.1.jar`.

The Maven artifact ID for the Java Client API is now `marklogic-client-api` instead of `java-client-api`. Update your build dependency configuration files, such as `pom.xml` or `build.gradle`, accordingly.

Logging Turned Off by Default

The Logback library is no longer included in the Java Client API distribution. Logging is now off by default. To re-enable logging, include Logback or another slf4j JAR in your classpath.

6.11.20. Node.js Client API Incompatibilities

MarkLogic 9 introduced some backward-incompatible changes to the Node.js Client API.

Changes to Return Value of documents.remove

The method `DatabaseClient.documents.remove` previously returned an object with both a "uri" property and a "uris" property that served the same purpose. The "uri" property has been removed. The return value now has the following form:

```
{ "uris":[uri1, uri2, ...], "removed":true }
```

The value of the "uris" property is now always an array. Previously, if you passed in just a single string as input, `remove` returned just the URI string, rather than an array of one item.

Transaction Creation Returns an Object by Default

Previously, `DatabaseClient.transactions.open` returned a transaction id string by default, and you could use the `withState` parameter to request a transaction object instead. The use of the string form was deprecated as of MarkLogic 8.

As of MarkLogic 9 and Node.js Client API v2.0.0, `DatabaseClient.transactions.open` defaults to returning a transaction object.

To force the previous behavior, set `withState` to `false` when creating a transaction. This setting is deprecated and will be removed in a future release.

Default Search Result Slice is Zero-Based

Previously, the slice clause on search (`queryBuilder.slice`) accepted a one-based starting position and a page length:

```
slice(oneBasedStart,PageLength)
```

As of MarkLogic 9 and Node.js Client API v2.0.0, `queryBuilder.slice` clause behaves like `Array.prototype.slice`. That is, it takes a zero-based starting position and the (zero-based) position after the last result to be retrieved. For example, the following slice call returns the first 5 results:

```
... .slice(0,5) ...
```

Also, you could previously use `slice(0)` to suppress the return of result documents and just retrieve an abbreviated summary. Now, you must include both the start and end positions for the same effect. For example: `slice(0,0)`.

To restore the legacy behavior, use `marklogic.setSliceMode`. Note, however, that this form is deprecated and will be removed in a later release.

NOTE

The semantics of valuesBuilder.slice are unchanged. This function still accepts as 1-based starting position and a page length.

6.11.21. Geospatial Region Accessors Can Now Return Double Values

The introduction of support for double precision coordinates in MarkLogic 9 means that some geospatial operations may return different results than in the past.

Previously, geospatial region accessor functions such as the XQuery function `cts:point-latitude` or the Server-Side JavaScript function `cts.pointLatitude` always returned float values. As of MarkLogic 9, these functions can return either single or double precision values, depending on the governing coordinate system. If you do not use a double precision coordinate system, you should not notice a difference.

The following functions are affected.

XQuery Function	JavaScript Function
`cts:point-latitude`	`cts.pointLatitude`
`cts:point-longitude`	`cts.pointLongitude`
`cts:box-west`	`cts.boxWest`
`cts:box-east`	`cts.boxEast`
`cts:box-south`	`cts.boxSouth`
`cts:box-north`	`cts.boxNorth`

For more details, see How Precision Affects Geospatial Operations in the *Search Developer's Guide*.

6.11.22. User-Defined Function Plugins Must Be Recompiled

The version of MarkLogic's C++ User-Defined Function (UDF) interface has been incremented to accommodate the following changes:

- The `marklogic::Point` class now accepts and returns longitude and latitude values as doubles instead of floats.
- Support for new types of UDF plugins in support of custom stemming and tokenization plugins.

You must recompile your UDF plugins for use with MarkLogic 9. UDFs from an earlier version of MarkLogic will not work in a mixed cluster prior to committing the new version. During a rolling upgrade, you must finish the upgrade and then recompile your UDFs for use with MarkLogic 9.

6.11.23. SLES 12 No Longer Supported

In MarkLogic 9, the SUSE Linux Enterprise Server operating system is not supported. If you are using this discontinued platform, you will need to migrate your environment to a supported platform. For details on supported platforms, see Supported Platforms in *Installing MarkLogic Server*.

6.11.24. Solaris No Longer Supported

In MarkLogic 9, the Solaris operating system is not supported. If you are using this discontinued platform, you will need to migrate your environment to a supported platform. For details on supported platforms, see Supported Platforms in *Installing MarkLogic Server*.

6.11.25. Nagios Plugin No Longer Supported

Support for the Nagios monitoring plugin has been discontinued in MarkLogic 9.

You can create your own MarkLogic monitoring integration using the MarkLogic REST Management API; for details see the Monitoring MarkLogic Guide.

MarkLogic does not endorse or support any particular 3rd party monitoring integrations. However, the MarkLogic open source community includes integrations with 3rd party monitoring platforms such as New Relic and App Dynamics. For more details see:

- New Relic: https://github.com/marklogic-community/newrelic-plugin
- App Dynamics: https://github.com/Appdynamics/marklogic-monitoring-extension (repository has been removed)

6.11.26. Application Builder and Information Studio No Longer Available

The Application Builder and Information Studio applications have been removed from MarkLogic. The info and infodev APIs remain, but they are deprecated; for details, see Section 7.2, "info and infodev APIs Deprecated" [56].

An Application Builder application deployed on MarkLogic 8 will continue to work after the upgrade to MarkLogic 9. We do not support any mechanism to support redeploying such an app against a MarkLogic 9 or later node. The methods and modules needed for this process have been removed.

Porting an Application Builder application running in MarkLogic 7 to MarkLogic 9 is not supported by MarkLogic Server. If you have an active maintenance contract, you can contact MarkLogic Technical Support for guidance in porting this application.

6.11.27. Admin Interface No Longer Selects a Default Schemas Database

As of MarkLogic 9, you can create a database without an associated schemas database. Earlier versions of MarkLogic required you to specify a schemas database when creating a new database. This change has no impact on users creating databases through the Admin API. However, this change affects the database creation page of the Admin Interface as follows:

The Admin Interface no longer automatically selects the pre-defined Schemas database as the schemas database when you create a database through the UI. Instead, you must explicitly select a schemas database or else you will create a database with no associated schemas database.

Depending on the uses to which you put the database, some operations might fail if there is no associated schemas database. For example, features such as temporal document management and template driven extraction require a schemas database to be associated with the content database.

6.11.28. Internal Security ON with External Security Object Behavior Change

This section describes some changes related to external security support. You should be aware of these changes, but they do not introduce incompatibilities or necessitate a change to your application.

In MarkLogic 9 and later, you can log into port 7001 on an LDAP account when “internal security” on appserver port 7001 is set to “true” or “false”. In previous versions of MarkLogic, you can only log into port 7001 on an LDAP account if “internal security” on appserver port 7001 is set to “false”.

In MarkLogic 9 and later, you can assign multiple external security objects to an App Server. When there are multiple external security objects assigned, a MarkLogic user is authenticated and assigned to an external security based on the order in which the external security objects are assigned. In previous version of MarkLogic, you could only assign one external security object to an App Server.

If internal security is enabled for an App Server, then when a user attempts to authenticate with MarkLogic, MarkLogic first checks to see if the user is in the security database. If so, then MarkLogic verifies the credentials against the security database. When this verification fails, the behavior of MarkLogic 8 and earlier versions differs from the behavior of MarkLogic 9 and later as follows:

- In MarkLogic 8 and earlier, if Security database verification fails, then the login attempt fails with an error.
- In MarkLogic 9 and later, if Security database verification fails, then MarkLogic attempts to authenticate the user against any external security objects assigned to the App Server. If there are no external security objects assigned or if the user cannot be authenticated against any assigned external security objects, then the login fails with an error.

Thus, when internal security is enabled, there can be cases where a login will succeed in MarkLogic 9 that would have failed with earlier versions. The behavior is unchanged if there are no external security objects assigned to the App Server or internal security is disabled.

6.11.29. REST Management API Changes in MarkLogic 9

In MarkLogic 9, the following REST Management API methods were deleted and their functionality was moved elsewhere. The following table lists the deleted methods and the new alternative.

Endpoint Deleted in 9.0	Alternative
`DELETE /manage/v2/credentials`	`DELETE:/manage/v2/credentials/properties`
`GET /manage/v2/credentials`	`GET:/manage/v2/credentials/properties`
`PUT /manage/v2/credentials`	`PUT:/manage/v2/credentials/properties`
`GET /manage/v2/databases/{id|name}/sub-databases`	`GET:/manage/v2/databases`
`POST /manage/v2/databases/{id|name}/sub-databases`	`POST:/manage/v2/databases`
`GET /manage/v2/databases/{id|name}/super-databases`	`GET:/manage/v2/databases`
`POST /manage/v2/databases/{id|name}/super-databases`	`POST:/manage/v2/databases`
`DELETE /manage/v2/databases/{sub-id-or-name}/super-databases/{super-id-or-name}`	`DELETE:/manage/v2/databases/{id|name}`
`GET /manage/v2/databases/{sub-id-or-name}/super-databases/{super-id-or-name}`	`GET:/manage/v2/databases/{id|name}/properties`

Endpoint Deleted in 9.0	Alternative
`DELETE /manage/v2/databases/{super-id-or-name}/sub-databases/{sub-id-or-name}`	`DELETE:/manage/v2/databases/{id|name}`
`GET /manage/v2/databases/{super-id-or-name}/sub-databases/{sub-id-or-name}`	`GET:/manage/v2/databases/{id|name}/properties`

6.11.30. Configuration Packaging Format Incompatibilities

Configuration packages created with the Packaging REST API and/or with the Configuration Packaging XQuery library module are not compatible with the configuration format introduced in MarkLogic release 9.0-5.

These configuration packages cannot be used with the CMA REST API and/or with the Configuration Management API XQuery/JavaScript library modules.

For more details, see the following sections:

- Configuration Management API (CMA) XQuery and JavaScript Libraries
- Configuration Management API (CMA) REST Endpoints
- Packaging API Removed

6.11.31. Configuration Management API (CMA) XQuery and JavaScript Libraries

MarkLogic 9.0-5 introduced library functions for configuration management.

The configuration management functions can be used to:

- retrieve a configuration of an individual resource, a set of resources, or a full cluster;
- generate a configuration from scenarios, such as High Availability (HA) scenario;
- apply a named configuration, overriding parameters and setting options.

Use the following functions for configuration management:

XQuery	Server-Side JavaScript
`cma:generate-config`	`cma.generateConfig`
`cma:apply-config`	`cma.applyConfig`

6.11.32. Configuration Management API (CMA) REST Endpoints

MarkLogic 9.0-5 introduced a REST API for configuration management: Configuration Management API (CMA).

The Configuration Management API is a RESTful API that allows retrieving, generating, and applying configurations for MarkLogic clusters, databases, and application servers.

Use the following endpoints for configuration management:

Endpoint	Description
`GET /manage/v3`	This endpoint enables retrieving configuration of an individual resource, a set of resources, or a full cluster. It also enables generating new configurations from scenarios.
`POST /manage/v3`	This endpoint enables applying named configurations to MarkLogic resources, overriding parameters and setting options. The configurations may be applied to an individual resource, a set of resources, or a full cluster.

For more details, see the *MarkLogic REST API Reference*.

6.11.33. Java Client API 4.1.1 Incompatibilities

Version 4.1.1 of the Java Client API introduced incompatibilities with earlier Java Client API versions.

Load Balancer Configuration for DMDSDK Jobs

This change does not affect you if you do not connect to MarkLogic through a load balancer or if you do not use the Data Movement SDK (DMSDK) feature of the Java Client API.

You must change the configuration of any `DatabaseClient` objects used to connect to MarkLogic through a load balancer on behalf of a Data Movement SDK job. The following changes apply:

- You must add a connection type parameter value of `GATEWAY` when configuring a DatabaseClient.
- Do not use a `FilteredForestConfiguration` to configure connections through a load balancer.

For more details, see Working with a Load Balancer in the *Java Application Developer's Guide*.

6.11.34. MarkLogic SQL ORDER-BY Keyword

Starting in MarkLogic 9.0-9, `ORDER-BY` ordering on NULL is set to `NULLS LAST`. This changes default behavior for nulls in query results and therefore may cause backwards incompatibility for certain queries (unless the `Optic Nulls Smallest On` trace event is turned on).

6.11.35. Changes to Accepted XML Character Set

As of MarkLogic 9.0-6, parsing of XML documents with an XML declaration that explicitly specifies XML version 1.1 (version="1.1") enforces the XML 1.1 character set. Consequently, you can now create content containing characters not accepted by XML 1.0.

Characters in the XML 1.1 “restricted character” ranges must be given as character entities. This enforcement applies to the following character ranges:

- 0x1-0x8
- 0xB-0xC
- 0xE-0x1F
- 0x7F-0x84
- 0x86-0x9F

The following character ranges that were previously disallowed are now accepted.

- 0x1-0x8
- 0xB-0xC
- 0xE-0x1F

Serializing content that contains characters allowed by XML 1.1 but not XML 1.0 can cause problems for client layers that cannot handle the XML 1.1 character set. You can control whether to serialize XML as version 1.0 or 1.1 by setting the output version in your App Server output options, using methods such as the following:

- Admin Interface: Set the output version to “1.0” under `Groups` > `yourGroup` > `App Servers` > `yourAppServer` > `Output Options`.
- Admin API: Call `admin:appserver-set-output-version` with a value of “1.0” for the $value parameter.
- REST Management API: Send a request to `PUT:/manage/v2/servers/{id|name}/properties` that sets the `output-version` property to “1.0”.

6.11.36. Minimum Required Version of HDP is 2.6

As of MarkLogic 9.0-5, if you are using the Hortonworks Data Platform (HDP) with the following MarkLogic technologies, you must use HDP v2.6.

- mlcp
- HDFS
- The MarkLogic Connector for Hadoop

6.11.37. Reindex Recommended for Geospatial Region Indexes

This change affects only database configuration that define a geospatial region index.

MarkLogic 9.0-5 introduces support for a “tolerance” option on geospatial region queries. As a consequence, if you use geospatial region indexes, your region queries might not be accurate until you reindex. This is applicable whether or not you take advantage of the new tolerance option.

6.11.38. Geospatial Region Query Results Might Differ

As of MarkLogic 9.0-5, MarkLogic uses the coordinate system *default* tolerance when evaluating a geospatial region query. Previously, MarkLogic used the coordinate system *minimum* tolerance.

Consequently, some region queries might not give the same results after upgrading to MarkLogic 9.0-5. Use the “tolerance” option to adjust your region queries, if necessary.

To learn more about tolerance, see Understanding Tolerance in the *Search Developer's Guide*.

6.11.39. return-query Option Output Format Change

This change only affects users of the REST, Java, and Node.js Client APIs who use the `return-query` query option and generate a search response summary in JSON.

In prior versions of MarkLogic, the `return-query` query option returned a serialized JSON structured query in the `query` property of the search response in JSON, but a serialized cts:query in an XML search response. As of MarkLogic 9.0-5, `return-query` always generates a serialized cts:query. The serialized JSON representation can be passed to `cts:query` or `cts.query` to reconstruct an in-memory query object on MarkLogic, just as you can with the XML serialization.

This change will not affect you if you do not use a JSON search response, do not use `return-query`, or do not have code that depends on the serialization produced by `return-query`.

6.11.40. Redaction: Deterministic Masking Values Differ

MarkLogic 9.0-4 introduced support for salting in the generation of masking values by the `mask-deterministic` redaction function. This means the same text redacted with `mask-deterministic` in earlier versions of MarkLogic will not produce the same masking value by default when redacted using MarkLogic 9.0-4.

To preserve the previous behavior, modify your `mask-deterministic` rules to set the `salt` and `extend-salt` options to an empty value.

For more details, see mask-deterministic in the *Application Developer's Guide*.

6.11.41. Incompatibilities Between 9.0-2 and 9.0-3

This section covers the incompatibilities that exist between MarkLogic 9.0-2 and MarkLogic 9.0-3.

Changes to Authentication Behavior with Client Certificate

MarkLogic Server 9.0-3 authenticates using both a client certificate and a username/password. This provides a greater level of security, by requiring that the user provide a client certificate that matches the specified user. In MarkLogic 9.0-2, if the password is incorrect, but the user has the correct client certificate, and the “ssl require client certificate” is false, authentication will succeed. In MarkLogic 9.0-3 if the password is incorrect, but the user has the correct client certificate and “ssl require client certificate is false”, authentication fails.

The “ssl require client certificate” is an appserver configuration. With the appserver, there are different authentication options: basic, digest, application-level, certificate, and kerberos-ticket. This change only

applies to basic, digest or application-level authentication. This behavior only happens when “ssl require client certificate” is set to false

In previous versions, MarkLogic would accept username/password forany internal user, once MarkLogic verified that certificate was signed by the selected CA (certificate authority). MarkLogic 9.0-1 and 9.0-2 authentication is restricted to an internal user with a matching common name (CN). In MarkLogic 9.0-3, the username does not need to match the common name (CN).

The behavior in MarkLogic 9.0-3 is the same as it is in MarkLogic 8.0. For more information, see Certificate in *Securing MarkLogic Server*.

XCC ContentSource.newSession Interface Change

The XCC method `ContentSource.newSession` has been extended to include a new overload that accepts a `char[]` value for the password parameter, rather than a `String`, to enable more secure handling of passwords.

As a consequence of this change, passing `null` in the password parameter of `newSession` now results in a compile time error because the compiler cannot determine the correct overload without type information. Use a typecast to disambiguate your call.

Document Digest Authorization Behavior Changed in 9.0-3

Prior to MarkLogic 9.0-3, the server did not check the nonce for digest authentication. In MarkLogic 9.0-3, the server checks the nonce, verifies that it is a valid nonce, and verifies that the URI in the Authorization header is same as the request URI.

1-click AMIs, new compatible CloudFormation, and additional upgrade procedures

To support 1-click deployment in AWS Marketplace, MarkLogic 9 AMIs have data volume pre-configured (device on `/dev/sdf`). To be compatible with 1-click AMIs, new CloudFormation templates are released on https://developer.marklogic.com/products/cloud/aws. To upgrade existing clusters, new 1-click compatible CloudFormation templates are required.

If custom templates or scripts are used, additional steps may be required to handle blank data volume that is part of the new marketplace AMIs. Please find more details about the upgrade procedure on Deploying MarkLogic on AWS.

map:new Retains Keys with Empty Values

As of MarkLogic 9.0-4, the `map:new` function retains any input key-value pair whose value is an empty sequence. Previous versions of MarkLogic 9 and versions of MarkLogic 8 prior to 8.0-7 discarded such key-value pairs.

For example:

```
map:keys(map:new(map:entry("noval",())))
(: Now returns "noval". Previously, returned an empty sequence. :)
```

The mlcp Option -tolerate_errors is Ignored

The `-tolerate_errors` option of the mlcp `import` command is deprecated. As of MarkLogic 9.0-2, mlcp ignores this option and always behaves as if `-tolerate_errors` is set to true. The option will be removed in a future release.

Changes to jsearch.facets Output Structure

In previous versions of MarkLogic, calling `jsearch.facets` always produced a JSON object for each facet, with object properties of the form `facetValue:`*count*. This structure prevented proper sorting of facet values.

As of MarkLogic 9.0-2, the structure of each facet is an array of arrays instead of a JSON object if and only if you include an explicit `orderBy` clause in your facet definition. If you do not use orderBy, the output is unchanged.

For more details, see Sorting Facet Values with OrderBy in the *Search Developer's Guide*.

Array Type is Preserved in x509 Certificate with Array-Valued Properties

In MarkLogic 9.0-1, if you use `xdmp.x509CertificateGenerate` to generate a certificate, and the configuration object includes array-valued properties, the array values were encoded as a single string. As of MarkLogic 9.0-2, the array type is preserved. This change applies to any Relative Distinguished Names (RDNs) within a Distinguished Name (DN), such as the `issuer` and `subject` DNs.

For example, in the following snippet, the `issuer.organizationName` property has an array value.

```
var certObj = {
  version: "2",
  serialNumber: "BA0195369CD6B679",
  issuer: {
    countryName: "US",
    stateOrProvinceName: "CA",
    localityName: "San Carlos",
    organizationName: ["MarkLogic", "Mark Logic"],
    organizationalUnitName: "Eng",
    emailAddress: "jdonner@marklogic.com",
    commonName: "JGD Certificate Authority",
  },...
};
var privateKey = ...;
xdmp.x509CertificatExtract(
  xdmp.x509CertificateGenerate(certObj, privateKey)
);
```

If you round trip the generated certificate through `xdmp.x509CertificateExtract`, you will see the following output for `issuer.organizationName` in MarkLogic 9.0-1 vs. MarkLogic 9.0-2.

```
// Marklogic 9.0-1
organizationName: "[\"MarkLogic\", \"Mark Logic\"]"
// MarkLogic 9.0-2 and later
organizationName: ["MarkLogic", "Mark Logic"]
```

If you do not have a certificate containing a multi-valued property, you will not notice any difference in behavior.

Node.js Client API: valuesBuilder.slice is Now Zero-Based

Previously, the slice clause on values queries (`valuesBuilder.slice`) accepted a one-based starting position and a page length:

```
slice(oneBasedStart,PageLength)
```

As Node.js Client API v2.0.3, the `valuesBuilder.slice` clause behaves like `Array.prototype.slice`. That is, it takes a zero-based starting position and the (zero-based) position after the last result to be retrieved. For example, the following slice call returns the first 5 results:

```
... .slice(0,5) ...
```

To restore the legacy behavior, use `marklogic.setSliceMode`. Note, however, that this form is deprecated and will be removed in a later release.

Changes to xdmp:update XQuery Prolog Option

Previously, setting the XQuery prolog option xdmp:update to “false” caused MarkLogic to automatically detect whether a module should be evaluated as an update or a query transaction.

As of MarkLogic 9.0-2, setting the option to “false” tells MarkLogic to treat the code as a query transaction. This could cause your program to get an error if you explicitly set the option to “false” and your code performs an update operation.

The new “auto” option value is equivalent to the previous behavior of “false”.

For related changes, see the following topics:

- Section 7.7, “xdmp:transaction-mode XQuery Prolog Option Deprecated” [57]
- Section 7.8, “Deprecation of transaction-mode Option to xdmp:eval” [57]

Java Client API 4.0.2 Ignores HttpClientConfigurator

As of version 4.0.2, the Java Client API uses OkHttp as its HTTP client for communicating with MarkLogic over HTTP. This change should be transparent to most applications, but imposes the following backward incompatibility on applications that customize their HTTP configuration:

Attaching a configurator based on `HttpClientConfigurator` to a `DatabaseClientFactory` object no longer has any effect on the HTTP configuration. Use the new `com.marklogic.client.extra.okhttpclient.OkHttpClientConfigurator` interface instead. The `HttpClientConfigurator` interface is deprecated and will be removed in a future release.

7. Planning for Future Upgrades

This section provides guidelines and warnings for preparing for changes expected in a future release, such as announcements of deprecated interfaces. You are not required to make changes related to the topics in this section at this time, but you should plan to do so in the future.

7.1. XQuery 0.9-ml Deprecated

The XQuery 0.9-ml dialect has been deprecated and will no longer be supported in a future release. MarkLogic recommends that you port any legacy 0.9-ml XQuery code to either enhanced XQuery dialect (1.0-ml) or strict XQuery dialect (1.0). For more information, see Porting 0.9-ml XQuery Code to Enhanced 1.0-ml in the *XQuery and XSLT Reference Guide*.

Template Driven Extraction (TDE) functions are also affected by this deprecation. The function cts:valid-index-path may throw error messages if you do not upgrade your code to XQuery dialect 1.0-ml or 1.0.

7.2. info and infodev APIs Deprecated

The XQuery library modules in the `info` and `infodev` namespaces are deprecated as of MarkLogic 9 and will be removed from the product in a future release. For example, the functions `info:ticket` and `infodev:ticket-create` are deprecated.

7.3. Annotated Query Output from search:parse Deprecated

The `search:parse` XQuery function and `search.parse` server-side JavaScript function can return an annotated `cts:query` if you pass in `"cts:annotated-query"` as the output format parameter value. As of MarkLogic 9, use of `"cts:annotated-query"` is deprecated. Support for this format will be removed in a future release.

If you currently use the annotated query output as an intermediate step in a transformation, you should use the structured query (`"search:query"`) output format instead. Runtime modification of queries is a primary use case for structured query. For more details, see Searching Using Structured Queries in the *Search Developer's Guide*.

If you currently use the annotated query output format to recover the original query text using `search:unparse`, you should cache the original query text yourself.

7.4. Search API Grammar Customization Deprecated

Customizing the string query grammar through the Search API `grammar` query option is deprecated. Support for this feature will be removed in a future release.

If your application currently relies on a Search API grammar customization, you should consider alternatives such as the following:

- xqysp (`http://github.com/mblakele/xqysp`) for XQuery.
- PEG.js (`http://pegjs.org/`) or Jison (`http://github.com/zaach/jison`) for Server-Side JavaScript.

7.5. Search API searchable-expression Deprecated

Due to security and performance considerations, beginning in MarkLogic 9.0-10, the `searchable-expression` property/element in query options is deprecated.

In addition, in 9.0-10 and moving forward, the `searchable-expression` requires the `eval-search-string` privilege.

Search API users should modify queries that use query options with searchable expressions to remove the `searchable-expression`. The replacement is as follows:

- For the parts of the searchable expression resolvable with indexes, use query clauses instead
- For the parts of the searchable expression resolvable only by scanning (such as regex predicates), the preferred approach is to remove them in favor of using indexes. Where necessary, filter the results in post-processing

7.6. The mlcp Option -tolerate_errors Deprecated

The `-tolerate_errors` option of the mlcp `import` command is deprecated (and ignored) as of MarkLogic 9.0-2. The option will be removed in a future release. mlcp now always tolerates errors.

7.7. xdmp:transaction-mode XQuery Prolog Option Deprecated

The XQuery prolog option `xdmp:transaction-mode` is deprecated as of MarkLogic 9.0-2. Use the `xdmp:commit` and `xdmp:update` prolog options instead.

Note that the new prolog options differ from `xdmp:transaction-mode` in that they affect only the transaction created after their declaration, whereas `xdmp:transaction-mode` settings persist across an entire session.

The following table illustrates the correspondence between the old and new option settings:

`xdmp:transaction-mode` value	Equivalent `xdmp:commit` and `xdmp:updateOption` Settings
`"auto"`	`declare option xdmp:commit "auto";` `declare option xdmp:update "auto";`
`"update-auto-commit"`	`declare option xdmp:commit "auto";` `declare option xdmp:update "true";`
`"update"`	`declare option xdmp:commit "explicit";` `declare option xdmp:update "true";`
`"query"`	`declare option xdmp:commit "explicit";` `declare option xdmp:update "false";`

Note that the default values for `xdmp:commit` and `xdmp:update` are both “auto”, so you do not need to set this value explicitly in most cases.

For more details, see xdmp:update and xdmp:commit in the *XQuery and XSLT Reference Guide*.

7.8. Deprecation of transaction-mode Option to xdmp:eval

The `transaction-mode` option of the `xdmp:eval` XQuery function and the `xdmp.eval` JavaScript function is deprecated as of MarkLogic 9.0-2. Use the `commit` and `update` options instead. For more details, see the function reference documentation for `xdmp:eval` (XQuery) and `xdmp.eval` (JavaScript).

This option deprecation (and alternative option settings) apply to the following functions:

XQuery	JavaScript
`xdmp:eval`	`xdmp.eval`
`xdmp:javascript-eval`	`xdmp.xqueryEval`
`xdmp:invoke`	`xdmp.invoke`
`xdmp:invoke-function`	`xdmp.invokeFunction`
`xdmp:spawn`	`xdmp.spawn`
`xdmp:spawn-function`	

The following table illustrates the correspondence between the old and new option settings:

transaction-mode Option Value	Equivalent commit and update Option Values
auto	commit: "auto" update: "auto"
update-auto-commit	commit: "auto" update: "true"
update	commit: "explicit" update: "true"
query	commit "explicit" update "false"

7.9. XCC Session.setTransactionMode is Deprecated

Use of `Session.setTransactionMode` to specify commit semantics and transaction type is deprecated as MarkLogic 9.0-2. This function will be removed in a future version. Use the new `Session.setAutoCommit` and `Session.setUpdate` methods instead.

The following table illustrates how to replace calls to `setTransactionMode` with equivalent calls to `setAutoCommit` and `setUpdate`:

If you call setTransactionMode with this value,	Then replace it with the following calls on the same Session object:
AUTO	setAutoCommit(true); setUpdate(Session.Update.AUTO);
MULTI_AUTO	setAutoCommit(false); setUpdate(Session.Update.AUTO);
UPDATE	setAutoCommit(false); setUpdate(Session.Update.TRUE);
QUERY	setAutoCommit(false); setUpdate(Session.Update.FALSE);
UPDATE_AUTO_COMMIT	setAutoCommit(true); setUpdate(Session.Update.TRUE);
QUERY_SINGLE_STATEMENT	setAutoCommit(true); setUpdate(Session.Update.FALSE);

7.10. Java Client API 4.0.2 Deprecations

Java Client API 4.0.2 introduces the following deprecations.

The `com.marklogic.client.extra.httpclient.HttpClientConfigurator` interface is deprecated and will be removed in a future release. Use the new `com.marklogic.client.extra.okhttpclient.OkHttpClientConfigurator` interface instead. Attaching a configurator based on `HttpClientConfigurator` to a `DatabaseClientFactory` object no longer has any effect on the HTTP configuration.

The single parameter version of `DatabaseClientFactory.SecurityContext.withSSLContext` has been deprecated and will be removed in a future release. Instead, use the new version that requires an `X509TrustManager` as its second parameter. This change affects all classes that implement `DatabaseClientFactory.SecurityContext`, such as `DatabaseClientFactory.CertificateAuthContext`, `DatabaseClientFactory.KerberosAuthContext`, and `DatabaseClientFactory.DigestAuthContext`.

7.11. Java Client API 4.0.4 Deprecations

Java Client API 4.0.4 deprecates some interfaces.

7.11.1. NamespacesManager Interface Deprecated

The Java Client API interfaces for configuration namespace bindings are deprecated as of version 4.0.4 and will be removed in a future release.

Instead, you should use the REST Management API to define namespace bindings for your App Server. For details, see the `namespaces` property of the payload for `PUT:/manage/v2/servers/{id|name}/properties`.

This deprecation notice affects the following components of the Java Client API:

- The `com.marklogic.client.admin.NamespacesManager` interface.
- The `com.marklogic.client.admin.ServerConfigurationManager.newNamespacesManager` method.

7.11.2. QueryBatcher.getQuerySuccessListeners Deprecated

The method `QueryBatcher.getQuerySuccessListeners` is deprecated as of Java Client API 4.0.0 and will be removed in a future release. Use `QueryBatcher.getUrisReaderListeners` instead.

Both methods do the same thing in the same way. You do not need to change anything but the method name.

7.12. REST Client API Namespace Configuration Deprecation

The REST Client API methods for configuring namespace bindings are deprecated as of MarkLogic 9.0-5 and will be removed in a future release.

Instead, you should use the REST Management API to define namespace bindings for your App Server. For details, see the `namespaces` property of the payload for `PUT:/manage/v2/servers/{id|name}/properties`.

This deprecation notice affects the following methods of the REST Client API:

- GET, POST, PUT and DELETE on `/v1/config/namespaces`
- GET, PUT, and DELETE on `/v1/config/namespaces/{prefix}`

7.13. CNTK Machine Learning Runtime Deprecated

The CNTK Machine Learning libraries and APIs are deprecated starting with MarkLogic release 10.0-4 and will be removed from the product in a future release. For more information, please see this Release Note from Microsoft.

7.14. Database Searches Using One or Two Characters Deprecated

Database searches using only one or two characters have been deprecated.

8. Other Notes

This section provides extra information about MarkLogic Server.

8.1. Memory and Disk Space Requirements

MarkLogic Server requires at least 2 GB of system memory.

The first time it runs, MarkLogic Server automatically configures itself to the amount of memory on the system, reserving as much as it can for its own use. If you need to change the default configuration, you can manually override these defaults at a later time using the Admin Interface.

MarkLogic recommends the following two guidelines for server sizing:

- Configure your server with 1 GB of physical memory for every 16 GB of source content you expect to manage.
- Configure your server with at least one CPU (or core) per 100 GB of source content.

Pragmatically, we recommend running most configurations with a minimum of two CPUs (or two cores).

MarkLogic Server requires 1.5 times the disk space of the total forest size. Specifically, each forest on a filesystem requires its filesystem to have at least 1.5 times the forest size in disk space (or, for each forest less than 48 GB, 3 times the forest size) when the `merge max size` database merge setting is set to the default of 48 GB. This translates to approximately 1.5 times the disk space of the source content after it is loaded. For example, if you plan on loading content that will result in a 200 GB database, reserve at least 300 GB of disk space. The disk space reserve is required for merges.

It is critical for swap space to be properly configured on your system according to the recommendations for your platform. For more details about memory, disk, and swap requirements, see Memory, Disk Space, and Swap Space Requirements in *Installing MarkLogic Server*.

* You need at least 2 times the `merge max size` of free space per forest, regardless of the forest size. Therefore, with the default `merge max size` of 48 GB, you need at least 96 GB of free space. Additionally, if your journals are not yet created, you need 2 times the journal size of free disk space (if the journal space is not yet allocated). Therefore, to be safe, you need (with the default `merge max size` and a 2G journal size) at least 100 GB of free space for each forest, no matter what size the forest is.

8.2. Compatibility with XQuery Specifications

MarkLogic implements the XQuery language, along with the functions and operators specified in the W3C XQuery 1.0 recommendations:

- http://www.w3.org/TR/xquery/
- http://www.w3.org/TR/xquery-operators/

Additionally, there is backwards compatibility with the May 2003 version of the XQuery 1.0 Draft specification used in MarkLogic Server 3.2 and previous versions. For details on the XQuery implementation in MarkLogic Server 4.1, including the three different dialects supported, see the *XQuery and XSLT Reference Guide*.

8.3. XQuery Extensions

Working within the W3C XQuery 1.0 Recommendation, MarkLogic has created a number of language extensions enabling key functionality not supported in the current release of the language specification. These extensions provide transactional update capabilities, assorted search and retrieval features, various data manipulation functions, and administrative tools.

The documentation for these extensions, as well as for the XQuery standard functions, is in the XQuery API documentation. Additional details about the XQuery dialects supported by MarkLogic can be found in the *XQuery and XSLT Reference Guide*.

8.4. Documentation

MarkLogic Server includes the following documentation:

Documentation	Description
Getting Started with Optic	Provides code examples in Optic, the SQL-like API that is MarkLogic's primary query language. It includes detailed explanations for querying text, values, graphs, geospatial regions, and metadata culminating in joining two different data models with a single query.
MarkLogic Server-Side JavaScript Function Reference	Provides API documentation for the Server-Side MarkLogic built-in extensions to the JavaScript standard functions.
MarkLogic XQuery and XSLT Function Reference	Provides API documentation for the MarkLogic built-in and module extensions to the XQuery standard functions, as well as API documentation for the W3C functions implemented in MarkLogic Server.
Getting Started with MarkLogic Server	Provides a quick, step-by-step overview of how to get up and running with MarkLogic Server.
Installing MarkLogic Server	Provides procedures for installing MarkLogic Server.
What's New in MarkLogic 11	Provides a summary of new features and upgrade compatibility.
Concepts Guide	Provides an overview of MarkLogic and conceptual information about the server architecture.
Application Developer's Guide	Provides procedures, methodologies, and conceptual information about general MarkLogic Server application development tasks.
Search Developer's Guide	Provides procedures, methodologies, and conceptual information about search-based application development tasks.
Node.js Application Developer's Guide	Provides procedures, methodologies, and conceptual information about developing MarkLogic Server applications using the Node.js Client API.
Java Application Developer's Guide	Provides procedures, methodologies, and conceptual information about developing MarkLogic Server applications using the Java API.
Developing with XCC	Provides an overview of the what you can do with the XCC libraries, examples of how to use XCC, and an overview of the sample applications included with XCC.
REST Application Developer's Guide	Provides information on MarkLogic Server administration and application development using the MarkLogic REST API.
Semantic Graph Developer's Guide	Provides procedures, methodologies, and conceptual information about semantic application development tasks and MarkLogic SPARQL and triple support.
Temporal Developer's Guide	Provides information on developing applications using MarkLogic bi-temporal features.
Entity Services Developer's Guide	Provides procedures, methodologies, and conceptual information about developing MarkLogic Applications using entity relationship modeling and the Entity Services API.
Reference Application Architecture Guide	Provides an overview of reference application architectures for multi-tiered applications built using MarkLogic as the database.
Administrating MarkLogic Server	Provides procedures for administrative tasks such as creating servers, creating databases, backing up databases, creating users, setting up your security policy, and so on.
Scripting Administrative Tasks Guide	Provides information on writing code to script various administrative tasks such as creating and modifying databases, App Servers, and so on.
Database Replication Guide	Provides information on database replication, useful for disaster recovery scenarios.
Flexible Replication Guide	Provides information on Flexible Replication, useful for information sharing from one database to another.
Monitoring MarkLogic Guide	Provides information on monitoring MarkLogic Server, including using the built-in monitoring tools and integrating with external tools such as HP Operations Manager.
JavaScript Reference Guide	Provides a language reference for the MarkLogic Server-Side JavaScript language. This guide includes MarkLogic-specific Object reference, but is not a comprehensive language reference.
XQuery and XSLT Reference Guide	Provides a condensed overview of the XQuery language, including information on the three XQuery dialects in MarkLogic Server. This guide does include some syntax information, although it is primarily intended as an introduction and quick reference to the language, not as a comprehensive reference.
mlcp User Guide	Provides a procedural guide that explains how to use MarkLogic Content Pump (mlcp), a command line tool to load content into MarkLogic, extract content from MarkLogic, or copy content between databases.

Documentation	Description
Content Processing Framework Guide	Provides an introduction to the Content Processing Framework and procedures for installing the default content processing framework.
Query Performance and Tuning Guide	Provides performance-related information that is useful to application developers and administrators.
Scalability, Availability, and Failover Guide	Provides information on large-scale system architecture, clustering, availability, and details on setting up shared-disk and local-disk failover.
Securing MarkLogic Server	Provides information on the role-based security model in MarkLogic Server.
MarkLogic Server on Amazon Web Services (AWS) Guide	Provides information about running MarkLogic Server in an EC2 environment.
Query Console User Guide	Provides step-by-step information on using Query Console, a tool to create and run arbitrary XQuery, JavaScript, SQL, and SPARQL code.
Loading Content Into MarkLogic Server	Provides procedures, methodologies, and conceptual information about loading content into MarkLogic Server. Includes an overview of the ingestion techniques available using XQuery, Java, REST, and the MarkLogic Content Pump (mlcp).
SQL Data Modeling Guide	Provides information on how to use MarkLogic's SQL interface, including the creation of relational schemas and views.
Messages and Codes Reference Guide	Provides a reference to MarkLogic Server and MarkLogic Application Services error and log messages.
Glossary, Copyright, and Support	Provides a glossary of terms as well as copyright and support information.
MarkLogic REST API Reference	Provides API documentation for the REST API.
Java Client API Documentation	Provides API documentation for the MarkLogic Java Client API.
Node.js Client API Documentation	Provides API documentation for the MarkLogic Node.js Client API.
XCC Javadoc API Documentation	Provides API documentation for the MarkLogic XML Contentbase Connector for Java API (XCC/J).
C++ UDF API Reference	Provides API documentation for the C++ User Defined Function (UDF) API.

XQuery language documentation is provided through the W3C working group drafts specified in Section 8.2, "Compatibility with XQuery Specifications" [60]. Sample code is provided through the demo server, which is automatically installed as part of the MarkLogic Server installation process. Additionally, there are many samples available on the MarkLogic developer website).

XQuery language extensions specific to MarkLogic Server are documented online in the MarkLogic XQuery and XSLT Function Reference. Example code snippets are provided as part of that documentation. The Admin Interface provides a large-scale example of complex XQuery programming, using many of the MarkLogic XQuery language extensions.

The Admin Interface includes built-in help screens that explain the purpose of its various controls and parameters.

Known bugs are documented online as we find them or as they are reported to us. See the MarkLogic support website (supported customers only) for more details.

8.5. Browser Requirements

The Admin Interface and other built-in GUI tools are supported on the following browsers:

- Chrome
- Microsoft Edge (Chromium Engine)

Other browser/platform combinations may work, but are not as thoroughly tested.

8.6. Security: Prevent Abuse of System Entity Expansion

Normal XML processing allows for external entities to be referenced and included in the parsed content of XML files. If you want to disable this processing, add the "Disable XML External Entities" trace event.

9. Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at `http://help.marklogic.com` to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the *Support Handbook* for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at `http://developer.marklogic.com`. For technical questions, we encourage you to ask your question on Stack Overflow.

10. Copyright

MarkLogic Server 11 and supporting products. Last updated: October, 2023.