Compatibility Changes in MongoDB 4.4

17-22 minutes

On this page

The following 4.4 changes can affect compatibility with older versions of MongoDB.

MongoDB removes the following command(s) and `mongo` shell helper(s):

MongoDB removes the following server parameter:

Removed Parameter	Description
`failIndexKeyTooLong`	MongoDB 4.4 removes the `failIndexKeyTooLong` parameter. This parameter was deprecated in 4.2 as MongoDB with

Removed Parameter	Description
	featureCompatibilityVersion (fCV) version 4.2+ no longer imposes an Index Key Limit.

Starting in version 4.4, the Windows MSI installer for both Community and Enterprise editions does not include the MongoDB Database Tools (`mongoimport`, `mongoexport`, etc). To download and install the MongoDB Database Tools on Windows, see Installing the MongoDB Database Tools.

If you were relying on the MongoDB 4.2 or previous MSI installer to install the Database Tools along with the MongoDB Server, you must now download the Database Tools separately.

Starting in Mongo 4.4, the rollback directory for a collection is named after the collection's UUID rather than the collection namespace; e.g.

```
<dbpath>/rollback/20f74796-d5ea-42f5-8c95-f79b39bad190/removed.2020-02-19T04-57-11.0.bson
```

For details, see Rollback Data.

The `replSetGetStatus` command and its `mongo` shell helper `rs.status()` removes the following deprecated fields from its output:

MongoDB 4.4 adds the `term` field to the replica set configuration document. Replica set members use `term` and `version` to achieve consensus on the "newest" replica configuration. Setting featureCompatibilityVersion (fCV) : "4.4" implicitly performs a `replSetReconfig` to add the `term` field to the configuration document and blocks until the new configuration propagates to a majority of replica set members. Similarly, downgrading to `fCV : "4.2"` implicitly performs a reconfiguration to remove the `term` field.

Starting in MongoDB 4.4, to run on a replica set member, the following operations require the member to be in PRIMARY or SECONDARY state.

- listDatabases
- listCollections
- listIndexes
- distinct
- dbStats
- collStats

If the member is in another state, such as STARTUP2, the operation errors.

In previous versions, the operations can also be run when the member is in STARTUP2. However, the operations wait until the member transitions to RECOVERING.

Starting in version 4.4, MongoDB deprecates specifying a settings.getLastErrorDefaults value other than the default of { w: 1, wtimeout: 0 }. MongoDB 4.4 honors any write concern value that you specify, however future MongoDB versions might not honor values other than the default. Instead, use the setDefaultRWConcern command to set the default read or write concern configuration for a replica set or sharded cluster.

Starting in MongoDB 4.4, find and findAndModify() projection can accept aggregation expressions and aggregation syntax.

With the use of aggregation expressions and syntax, including the use of literals and aggregation variables, if you specify a literal (other than a number or a boolean) for the projection field value, the field is projected with the new value.

For example, consider a collection inventory with documents that contain a status field:

```
db.inventory.insertOne( { _id: 1, item:
```

```
"postcard", status: "A", instock: [ { warehouse:
"B", qty: 15 }, { warehouse: "C", qty: 35 } ] })
```

Starting in MongoDB, 4.4, the following operation projects the fields `status` and `instock` with new values instead of their current value:

```
db.inventory.find(
   { status: "A" },
   { status: "Active", instock: ["blue",
"crimson"] }
)
```

That is, the operation returns the following document:

```
{ "_id" : 1, "status" : "Active", "instock" : [
"blue", "crimson" ] }
```

In previous versions, any specification value (with the exception of zero/false value or the previously unsupported document value) is treated as `true` to indicate the inclusion of the field with its current value. That is, in earlier versions, the previous operation returns a document with the `status` and `instock` fields with their current values:

```
{ "_id" : 1, "status" : "A", "instock" : [ {
"warehouse" : "B", "qty" : 15 }, { "warehouse" :
"C", "qty" : 35 } ] }
```

Starting in MongoDB 4.4, regardless of the ordering of the fields in the document, the `$elemMatch` projection of an existing field returns the field after the other existing field inclusions.

For example, consider a `players` collection with the following document:

```
db.players.insertOne( {
```

```
   name: "player1",
   games: [ { game: "abc", score: 8 }, { game: "xyz", score: 5 } ],
   joined: new Date("2020-01-01"),
   lastLogin: new Date("2020-05-01")
} )
```

In version 4.4+, the following projection returns the `games` field after the other existing fields included in the projection even though in the document, the field is listed before `joined` and `lastLogin` fields:

```
db.players.find( {}, { games: { $elemMatch: { score: { $gt: 5 } } }, joined: 1, lastLogin: 1 } )
```

That is, the operation returns the following document:

```
{
   "_id" : ObjectId("5edef64a1c099fff6b033977"),
   "joined" : ISODate("2020-01-01T00:00:00Z"),
   "lastLogin" : ISODate("2020-05-01T00:00:00Z"),
   "games" : [ { "game" : "abc", "score" : 8 } ]
}
```

In version 4.2 and earlier, the [$elemMatch] projection of an existing field upholds the ordering in the document:

```
{
```

```
  "_id" : ObjectId("5edef91e76ddff7d92f118e1"),
  "games" : [ { "game" : "abc", "score" : 8 } ],
  "joined" : ISODate("2020-01-01T00:00:00Z"),
  "lastLogin" : ISODate("2020-05-01T00:00:00Z")
}
```

Starting in MongoDB 4.4, the `$slice` projection of an array in an nested document no longer returns the other fields in the nested document when the projection is part of an inclusion projection.

For example, consider a collection `inventory` with documents that contain a `size` field:

```
{ item: "socks", qty: 100, details: { colors: [ "blue", "red" ], sizes: [ "S", "M", "L"] } }
```

Starting in MongoDB 4.4, the following operation projects the `_id` field (by default), the `qty` field, and the `details` field with just the specified slice of the `colors` array:

```
db.inventory.find( { }, { qty: 1, "details.colors": { $slice: 1 } } )
```

That is, the operation returns the following document:

```
{ "_id" : ObjectId("5ee92a6ec644acb6d13eedb1"), "qty" : 100, "details" : { "colors" : [ "blue" ] } }
```

If the `$slice` projection is part of an exclusion projection, the operation continues to return the other fields in the nested document. That is, the following projection is an exclusion projection. The projection excludes the `_id` field and the elements in the `colors` array that fall outside the specified slice and returns all other fields.

```
db.inventory.find( { }, { _id: 0,
"details.colors": { $slice: 1 } } )
```

```
{ "item" : "socks", "qty" : 100, "details" : {
"colors" : [ "blue" ], "sizes" : [ "S", "M", "L"
] } }
```

The `$slice` projection by itself is considered an exclusion.

In previous versions, the `$slice` projection also include the other fields in the nested document regardless of whether the projection is an inclusion or an exclusion.

Starting in MongoDB 4.4, it is illegal to project an embedded document with any of the embedded document's fields.

For example, consider a collection `inventory` with documents that contain a `size` field:

```
{ ..., size: { h: 10, w: 15.25, uom: "cm" }, ...
}
```

Starting in MongoDB 4.4, the following operation fails with a `Path collision` error because it attempts to project both `size` document and the `size.uom` field:

```
db.inventory.find( {}, { size: 1, "size.uom": 1 }
)  // Invalid starting in 4.4
```

In previous versions, lattermost projection between the embedded documents and its fields determines the projection:

- If the projection of the embedded document comes after any and all projections of its fields, MongoDB projects the embedded document. For example, the projection document `{ "size.uom": 1, size: 1 }` produces the same result as the projection document `{ size: 1 }`.
- If the projection of the embedded document comes before the projection any of its fields, MongoDB projects the specified field or

fields. For example, the projection document `{ "size.uom": 1, size: 1, "size.h": 1 }` produces the same result as the projection document `{ "size.uom": 1, "size.h": 1 }`.

Starting in MongoDB 4.4, find and `findAndModify()` projection cannot contain both a `$slice` of an array and a field embedded in the array.

For example, consider a collection `inventory` that contains an array field `instock`:

```
{ ..., instock: [ { warehouse: "A", qty: 35 }, { warehouse: "B", qty: 15 }, { warehouse: "C", qty: 35 } ], ... }
```

Starting in MongoDB 4.4, the following operation fails with a `Path collision` error:

```
db.inventory.find( {}, { "instock": { $slice: 1 }, "instock.warehouse": 0 } ) // Invalid starting in 4.4
```

In previous versions, the projection applies both projections and returns the first element (`$slice: 1`) in the `instock` array but suppresses the `warehouse` field in the projected element. Starting in MongoDB 4.4, to achieve the same result, use the `db.collection.aggregate()` method with two separate `$project` stages.

Starting in MongoDB 4.4, the find and `findAndModify()` projection cannot project a field that starts with $ with the exception of the DBRef fields.

For example, starting in MongoDB 4.4, the following operation is invalid:

```
db.inventory.find( {}, { "$instock.warehouse": 0, "$item": 0, "detail.$price": 1 } ) // Invalid starting in 4.4
```

In earlier version, MongoDB ignores the $-prefixed field projections.

Starting in MongoDB 4.4, the $ projection operator can only appear at the end of the field path; e.g. `"field.$"` or `"fieldA.fieldB.$"`.

For example, starting in MongoDB 4.4, the following operation is invalid:

```
db.inventory.find( { }, { "instock.$.qty": 1 } )
// Invalid starting in 4.4
```

To resolve, remove the component of the field path that follows the $ projection operator.

In previous versions, MongoDB ignores the part of the path that follows the $; i.e. the projection is treated as `"instock.$"`.

Starting in MongoDB 4.4, find and `findAndModify()` projection cannot include `$slice` projection expression as part of a $ projection expression.

For example, starting in MongoDB 4.4, the following operation is invalid:

```
db.inventory.find( { "instock.qty": { $gt: 25 } }, { "instock.$": { $slice: 1 } } ) // Invalid starting in 4.4
```

In previous versions, MongoDB returns the first element (`instock.$`) in the `instock` array that matches the query condition; i.e. the positional projection `"instock.$"` takes precedence and the `$slice:1` is a no-op. The `"instock.$": { $slice: 1 }` does not exclude any other document field.

Starting in MongoDB 4.4, find and `findAndModify()` projection cannot include a projection of an empty field name.

For example, starting in MongoDB 4.4, the following operation is invalid:

```
db.inventory.find( { }, { "": 0 } ) // Invalid
```

```
starting in 4.4
```

In previous versions, MongoDB treats the inclusion/exclusion of the empty field as it would the projection of non-existing fields.

Starting in MongoDB 4.4, you must specify the `$text` operator in the query predicate of the `db.collection.find()` operations to use `{ $meta: "textScore" }` expression in the projection or sort. For example:

```
db.articles.find(
   { $text: { $search: "cake" } },
   { score: { $meta: "textScore" } }
);

db.articles.find(
   { $text: { $search: "cake" } },
   { score: { $meta: "textScore" } }
).sort( { score: {  $meta: "textScore" } } );
```

If you do not specify the `$text` operator in the query predicate, the operation fails. For example, the following operations are invalid starting in MongoDB 4.4:

```
db.articles.find(
   { },
   { score: { $meta: "textScore" } }
)

db.articles.find(
   { },
   { score: { $meta: "textScore" } }
```

```
).sort( { score: {  $meta: "textScore" } } );
```

Starting in MongoDB 4.4, the `sort()` method now uses the same sort algorithm as the `$sort` aggregation stage. With this change, queries which perform a `sort()` on fields that contain duplicate values are much more likely to result in inconsistent sort orders for those values.

To guarantee sort consistency when using `sort()` on duplicate values, include an additional field in your sort that contains exclusively unique values.

This can be accomplished easily by adding the `_id` field to your sort.

See Sort Consistency for more information.

Starting in MongoDB 4.4, `mapReduce` removes the `counts` field from its output.

In earlier versions, the command includes a `counts` field in its output. For example:

```
"counts" : {
   "input" : 4,
   "emit" : 4,
   "reduce" : 1,
   "output" : 2
},
```

Starting in MongoDB 4.4, the `map` function no longer restricts the size of each `emit()` output to a half of MongoDB's maximum BSON document size.

In earlier versions, a single emit can only hold half of MongoDB's maximum BSON document size

`mapReduce` no longer supports the deprecated BSON type

JavaScript code with scope (BSON type 15) for its functions. The `map`, `reduce`, and `finalize` functions must be either BSON type String (BSON type 2) or BSON type JavaScript (BSON type 13). To pass constant values which will be accessible in the `map`, `reduce`, and `finalize` functions, use the `scope` parameter.

The use of JavaScript code with scope for the `mapReduce` functions has been deprecated since version 4.2.1.

Starting in MongoDB 4.4, `mongod` / `mongos` instances now output all log messages in structured JSON format. This includes log output sent to the *file*, *syslog*, and *stdout* (standard out) log destinations, as well as the output of the `getLog` command.

Previously, log entries were output as plaintext.

If you have existing log parsing utilities, or use a log ingestion service, you may need to reconfigure these tools for the new structured logging format with MongoDB 4.4.

See Log Messages for a detailed examination of the new structured logging format, including examples of log parsing using the new log structure.

Starting in MongoDB 4.4, the `getLog` command no longer accepts the `rs` value, as this categorization of message type has been deprecated. Instead, log messages are now always identified by their component, including **REPL** for replication messages.

See Filtering by Component for log parsing examples that filter on the component field.

With the transition to structured JSON logging, the `ctime` timestamp format is no longer supported. The following configuration options no longer accept `ctime` as a valid parameter:

- `systemLog.timeStampFormat`
- `mongod --timeStampFormat`
- `mongos --timeStampFormat`

Use the `iso8601-local` (default) or `iso8601-utc` timestamp formats instead.

With the transition to structured JSON logging, the `maxLogSizeKB` server parameter now truncates any individual attributes in a log entry that exceed the specified limit. Previously, this parameter would truncate the entire log entry.

In addition:

- `maxLogSizeKB` now accepts a value of `0`, which disables truncation entirely.
- `maxLogSizeKB` no longer accepts negative values.

  See log message truncation for more information.
- MongoDB 4.4 removes support for gperftools cpu profiler. As part of this change, the `hostManager` no longer provides `cpuProfiler` privilege action on the cluster.
- The parameter `ldapConnectionPoolMaximumConnectionsPerHost` now has a default value of 2. In previous versions, the default is unset.
- `serverStatus` returns `flowControl.locksPerKiloOp` instead of `flowControl.locksPerOp`.
- The `$dateFromParts` expression operator now supports a value range of 1–9999 for the `year` and `isoWeekYear` fields. In previous versions, the supported value range for these fields was `0`–9999.
- The `listIndexes` and the `mongo` shell helper method `db.collection.getIndexes()` no longer returns the namespace `ns` field in the index specification documents.
- MongoDB 4.4 removes the `--noIndexBuildRetry` command-line option and the corresponding `storage.indexBuildRetry` option.
- `mongos` now logs an error if you pass an empty `writeConcern`

value i.e. `writeConcern: {}` to a command that does not support write concerns. In earlier versions, `mongos` ignores an empty `writeConcern` value for these commands.

- The `force` option with the `compact` command is no no longer a boolean. `force: true` and `force: false` are deprecated, and will result in an error.

  The `mongo` method `db.collection.validate()` no longer accepts just a boolean parameter.

  That is, the method no longer accepts `db.collection.validate(<boolean>)` as a shorthand for `db.collection.validate({full: <boolean>})`:

Instead of	Use
`db.collection.validate(true)`	`db.collection.validate({ full: true })`

`db.collection.validate(false)`	`db.collection.validate()`
	-or-
	`db.collection.validate({ full: false })`

  Starting in MongoDB 4.4, full validation on the `oplog` for WiredTiger skips the more thorough check. The `validate.warnings` includes a notice of the behavior.

- `dbStats` command no longer returns the obsolete MMAPv1 `numExtents` field.

- `replSetGetStatus` command no longer returns the obsolete MMAPv1 field `replSetGetStatus.initialSyncStatus.fetchedMissingDocs` in its output.

- `fsync` command no longer accepts the obsolete MMAPv1 field `async` as an option.

MongoDB 4.4 deprecates the geoHaystack index and the `geoSearch` command. Use a 2d index with `$geoNear` or `$geoWithin` instead.

Starting in MongoDB 4.4:

- `$where` no longer supports the deprecated BSON type JavaScript code with scope (BSON type 15). The `$where` operator only supports BSON type String (BSON type 2) or BSON type JavaScript (BSON type 13).
- `mapReduce` no longer supports the deprecated BSON type JavaScript code with scope (BSON type 15) for its functions. The `map`, `reduce`, and `finalize` functions must be BSON type String (BSON type 2) or BSON type JavaScript (BSON type 13). To pass constant values which will be accessible in the `map`, `reduce`, and `finalize` functions, use the `scope` parameter.

  The use of BSON type JavaScript code with scope for `$where` and the `mapReduce` functions has been deprecated since MongoDB 4.2.1.

  MongoDB 4.4 deprecates the following sharding commands:

- `shardConnPoolStats` (use `connPoolStats` instead)
- `unsetSharding`

  The WiredTiger lookaside table (LAS) cache overflow file no longer exists starting in MongoDB 4.4. As such, MongoDB 4.4 deprecates the following options and parameter for the (LAS) cache overflow file limit; these options and parameter have no effect starting in MongoDB 4.4:

- `storage.wiredTiger.engineConfig.maxCacheOverflowFileSizeG` configuration file option
- `--wiredTigerMaxCacheOverflowFileSizeGB` command-line option
- `wiredTigerMaxCacheOverflowSizeGB` parameter

Some features in 4.4 require not just the 4.4 binaries but the featureCompatibilityVersion (fCV) set to 4.4. These features include:

- Raises the Namespace Length limit for MongoDB versions with fCV set to 4.4+.
- Creation of Compound Hashed Indexes requires fCV set to 4.4+.