Command Logging and Monitoring

- Status: Accepted
- Minimum Server Version: 2.4

Specification

The command logging and monitoring specification defines a set of behaviour in the drivers for providing runtime information about commands in log messages as well as in events which users can consume programmatically, either directly or by integrating with third-party APM libraries.

Definitions

META

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Terms

Document

The term `Document` refers to the implementation in the driver's language of a BSON document.

Guidance

Documentation

The documentation provided in code below is merely for driver authors and SHOULD NOT be taken as required documentation for the driver.

Messages and Events

All drivers MUST implement the specified event types as well as log messages.

Implementation details are noted in the comments when a specific implementation is required. Within each event and log message, all properties are REQUIRED unless noted otherwise.

Naming

All drivers MUST name types, properties, and log message values as defined in the following sections.

Exceptions to this rule are noted in the appropriate section. Class and interface names may vary according to the driver and language best practices.

Publishing & Subscribing

The driver SHOULD publish events in a manner that is standard to the driver's language publish/subscribe patterns and is not strictly mandated in this specification.

Similarly, as described in the logging specification the driver SHOULD emit log messages in a manner that is standard for the language.

Guarantees

The driver MUST guarantee that every `CommandStartedEvent` has either a correlating `CommandSucceededEvent` or `CommandFailedEvent`, and that every "command started" log message has either a correlating "command succeeded" log message or "command failed" log message.

The driver MUST guarantee that the `requestId` of the `CommandStartedEvent` and the corresponding `CommandSucceededEvent` or `CommandFailedEvent` is the same, and that the `requestId` of the "command started" log message and the corresponding "command succeeded" or "command failed" log message is the same.

Unacknowledged/Acknowledged Writes

Unacknowledged writes must provide a `CommandSucceededEvent` and a "command succeeded" log message with a `{ ok: 1 }` reply.

A non-default write concern MUST be included in the published command. The default write concern is not required to be included.

Succeeded or Failed

Commands that executed on the server and return a status of `{ ok: 1.0 }` are considered successful commands and MUST generate a `CommandSucceededEvent` and "command succeeded" log message. Commands that have write errors are included since the actual command did succeed, only writes failed.

Error Handling

If an exception occurs while sending the operation to the server, the driver MUST generate a `CommandFailedEvent` and "command failed" log message with the exception or message, and re-raise the exception.

Bulk Writes

This specification defines the monitoring and logging of individual commands and in that respect MUST generate events and log messages for each command a bulk write executes. Each of these commands, however, must be linked together via the same `operationId`.

Implementation Notes

When a driver sends an OP_MSG with a document sequence, it MUST include the document sequence as a BSON array in `CommandStartedEvent.command`. The array's field name MUST be the OP_MSG sequence identifier. For example, if the driver sends an `update` command using OP_MSG, and sends a document sequence as a separate section of payload type 1 with identifier `updates`, the driver MUST include the documents as a BSON array in `CommandStartedEvent.command` with field name `updates`.

See "Why are document sequences included as BSON arrays?" in the rationale.

Rationale

1. Why are commands with `{ ok: 1 }` treated as successful and `{ ok: 0 }` as failed?

The specification is consistent with what the server deems as a successful or failed command and reports this as so. This also allows for server changes around this behaviour in the future to require no change in the drivers to continue to be compliant.

The command listener API is responsible only for receiving and handling events sent from the lowest level of the driver, and is only about informing listeners about what commands are sent and what replies are received. As such, it would be innappropiate at this level for a driver to execute custom logic around particular commands to determine what failure or success means for a particular command. Implementers of the API are free to handle these events as they see fit, which may include code that further interprets replies to specific commands based on the presence or absence of other fields in the reply beyond the `ok` field.

2. Why are document sequences included as BSON arrays?

The OP_MSG wire protocol was introduced in MongoDB 3.6, with document sequences as an optimization for bulk writes. We have chosen to represent these OP_MSGs as single command documents for now, until a need for a more accurate (and perhaps better-performing) command monitoring API for document sequences has been demonstrated.

3. Why is BSON serialization and deserialization optional to include in durations?

Different drivers will serialize and deserialize BSON at different levels of the driver architecture. For example, some parts of a command (e.g. inserted document structs) could be pre-encoded early into a "raw" BSON form and the final command with late additions like a session ID could encoded just before putting it on the wire.

Rather than specify a duration rule that would be hard to satisfy consistently, we allow duration to include

BSON serialization/deserialization or not based on the architecture needs of each driver.

Security

Some commands and replies will contain sensitive data relating to authentication.

In order to not risk leaking this data to external sources or logs, for these commands:

- The "command" field in `CommandStartedEvent` and "command started" log messages MUST be replaced with an empty BSON document.
- The "reply" field in `CommandSucceededEvent` and "command succeeded" log messages MUST be replaced with an empty BSON document.
- If the error is a server-side error, the "failure" field in `CommandFailedEvent` and "command failed" log messages MUST have all fields besides the following redacted:
  - `code`
  - `codeName`
  - `errorLabels`

  The exact implementation of redaction is flexible depending on the type the driver uses to represent a failure in these events and log messages. For example, a driver could choose to set all properties besides these on an error object to null. Alternatively, a driver that uses strings to represent failures could replace relevant portions of the string with "REDACTED".

The list of sensitive commands is as follows:

Command
`authenticate`
`saslStart`
`saslContinue`
`getnonce`
`createUser`
`updateUser`
`copydbgetnonce`
`copydbsaslstart`
`copydb`
`hello` (or legacy hello) when `speculativeAuthenticate` is present

See the MongoDB Handshake spec for more information on `hello` and legacy hello. Note that legacy hello has two different letter casings that must be taken into account. See the previously mentioned MongoDB Handshake spec for details.

Events API

See the Load Balancer Specification for details on the `serviceId` field.

```
interface CommandStartedEvent {

  /**
   * Returns the command.
   */
  command: Document;

  /**
   * Returns the database name.
   */
  databaseName: String;

  /**
   * Returns the command name.
   */
  commandName: String;

  /**
   * Returns the driver generated request id.
   */
  requestId: Int64;

  /**
   * Returns the driver generated operation id. This is used to link
events together such
   * as bulk write operations. OPTIONAL.
   */
  operationId: Int64;

  /**
   * Returns the connection id for the command. For languages that do not
have this,
   * this MUST return the driver equivalent which MUST include the server
address and port.
   * The name of this field is flexible to match the object that is
returned from the driver.
   */
  connectionId: ConnectionId;
```

```
  /**
   * Returns the server connection id for the command. The server
connection id is distinct from
   * the connection id and is returned by the hello or legacy hello
response as "connectionId"
   * from the server on 4.2+. Drivers MUST use an Int64 to represent the
server connection ID value.
   */
  serverConnectionId: Optional<Int64>;

  /**
   * Returns the service id for the command when the driver is in load
balancer mode.
   * For drivers that wish to include this in their ConnectionId object,
this field is
   * optional.
   */
  serviceId: Optional<ObjectId>;
}

interface CommandSucceededEvent {

  /**
   * Returns the execution time of the event in the highest possible
resolution for the platform.
   * The calculated value MUST be the time to send the message and receive
the reply from the server
   * and MAY include BSON serialization and/or deserialization. The name
can imply the units in which the
   * value is returned, i.e. durationMS, durationNanos.
   */
  duration: Int64;

  /**
   * Returns the command reply.
   */
  reply: Document;

  /**
   * Returns the command name.
   */
  commandName: String;

  /**
   * Returns the database name.
   */
  databaseName: String;
```

```
  /**
   * Returns the driver generated request id.
   */
  requestId: Int64;

  /**
   * Returns the driver generated operation id. This is used to link
events together such
   * as bulk write operations. OPTIONAL.
   */
  operationId: Int64;

  /**
   * Returns the connection id for the command. For languages that do not
have this,
   * this MUST return the driver equivalent which MUST include the server
address and port.
   * The name of this field is flexible to match the object that is
returned from the driver.
   */
  connectionId: ConnectionId;

  /**
   * Returns the server connection id for the command. The server
connection id is distinct from
   * the connection id and is returned by the hello or legacy hello
response as "connectionId"
   * from the server on 4.2+. Drivers MUST use an Int64 to represent the
server connection ID value.
   */
  serverConnectionId: Optional<Int64>;

  /**
   * Returns the service id for the command when the driver is in load
balancer mode.
   * For drivers that wish to include this in their ConnectionId object,
this field is
   * optional.
   */
  serviceId: Optional<ObjectId>;
}

interface CommandFailedEvent {

  /**
   * Returns the execution time of the event in the highest possible
resolution for the platform.
```

```
   * The calculated value MUST be the time to send the message and receive
the reply from the server
   * and MAY include BSON serialization and/or deserialization. The name
can imply the units in which the
   * value is returned, i.e. durationMS, durationNanos.
   */
  duration: Int64;

  /**
   * Returns the command name.
   */
  commandName: String;

  /**
   * Returns the database name.
   */
  databaseName: String;

  /**
   * Returns the failure. Based on the language, this SHOULD be a message
string, exception
   * object, or error document.
   */
  failure: String,Exception,Document;

  /**
   * Returns the client generated request id.
   */
  requestId: Int64;

  /**
   * Returns the driver generated operation id. This is used to link
events together such
   * as bulk write operations. OPTIONAL.
   */
  operationId: Int64;

  /**
   * Returns the connection id for the command. For languages that do not
have this,
   * this MUST return the driver equivalent which MUST include the server
address and port.
   * The name of this field is flexible to match the object that is
returned from the driver.
   */
  connectionId: ConnectionId;

  /**
```

```
    * Returns the server connection id for the command. The server
connection id is distinct from
    * the connection id and is returned by the hello or legacy hello
response as "connectionId"
    * from the server on 4.2+. Drivers MUST use an Int64 to represent the
server connection ID value.
    */
  serverConnectionId: Optional<Int64>;

  /**
    * Returns the service id for the command when the driver is in load
balancer mode.
    * For drivers that wish to include this in their ConnectionId object,
this field is
    * optional.
    */
  serviceId: Optional<ObjectId>;
}
```

Log Messages

Please refer to the logging specification for details on logging implementations in general, including log levels, log components, and structured versus unstructured logging.

Drivers MUST support logging of command information via the following types of log messages. These messages MUST be logged at `Debug` level and use the `command` log component.

The log messages are intended to match the information contained in the events above. Drivers MAY implement command logging support via an event subscriber if it is convenient to do so.

The types used in the structured message definitions below are demonstrative, and drivers MAY use similar types instead so long as the information is present (e.g. a double instead of an integer, or a string instead of an integer if the structured logging framework does not support numeric types.)

Drivers MUST not emit command log messages for commands issued as part of the handshake with the server, or heartbeat commands issued by server monitors.

Common Fields

The following key-value pairs MUST be included in all command messages:

Key	Suggested Type	Value
commandName	String	The command name.

databaseName	String	The database name.
requestId	Int	The driver-generated request ID.
operationId	Int	The driver-generated operation ID. Optional; only present if the driver generated operation IDs and this command has one.
driverConnectionId	Int64	The driver's ID for the connection used for the command. Note this is NOT the same as `CommandStartedEvent.connectionId` defined above, but refers to the `connectionId` defined in the connection monitoring and pooling specification. Unlike `CommandStartedEvent.connectionId` this field MUST NOT contain the host/port; that information MUST be in the following fields, `serverHost` and `serverPort`. This field is optional for drivers that do not implement CMAP if they do have an equivalent concept of a connection ID.
serverHost	String	The hostname or IP address for the server the command is being run on.
serverPort	Int	The port for the server the command is being run on. Optional; not present for Unix domain sockets. When the user does not specify a port and the default (27017) is used, the driver SHOULD include it here.
serverConnectionId	Int64	The server's ID for the connection used for the command. Optional; only present for server versions 4.2+. NOTE: Existing drivers may represent this as an Int32 already. For strongly-typed languages, you may have to introduce a new Int64 field and deprecate the old Int32 field. The next major version should remove the old Int32 field.
serviceId	String	The hex string representation of the service ID for the command. Optional; only present when the driver is in load balancer mode.

Command Started Message

In addition to the common fields, command started messages MUST contain the following key-value pairs:

Key	Suggested Type	Value
message	String	"Command started"

command	String	Relaxed extJSON representation of the command. This document MUST be truncated appropriately according to rules defined in the logging specification, and MUST be replaced with an empty document "{ }" if the command is considered sensitive.

The unstructured form SHOULD be as follows, using the values defined in the structured format above to fill in placeholders as appropriate:

> Command "{{commandName}}" started on database "{{databaseName}}" using a connection with driver-generated ID {{driverConnectionId}} and server-generated ID {{serverConnectionId}} to {{serverHost}}:{{serverPort}} with service ID {{serviceId}}. The requestID is {{requestId}} and the operation ID is {{operationId}}. Command: {{command}}

Command Succeeded Message

In addition to the common fields, command succeeded messages MUST contain the following key-value pairs:

Key	Suggested Type	Value
message	String	"Command succeeded"
durationMS	Int32/Int64/Double	The execution time for the command in milliseconds. The calculated value MUST be the time to send the message and receive the reply from the server and MAY include BSON serialization and/or deserialization.
reply	String	Relaxed extJSON representation of the reply. This document MUST be truncated appropriately according to rules defined in the logging specification, and MUST be replaced with an empty document "{ }" if the command is considered sensitive.

The unstructured form SHOULD be as follows, using the values defined in the structured format above to fill in placeholders as appropriate:

> Command "{{commandName}}" succeeded on database "{{databaseName}}" in {{durationMS}} ms using a connection with driver-generated ID {{driverConnectionId}} and server-generated ID {{serverConnectionId}} to {{serverHost}}:{{serverPort}} with service ID {{serviceId}}. The requestID is {{requestId}} and the operation ID is {{operationId}}. Command reply: {{command}}

Command Failed Message

In addition to the common fields, command failed messages MUST contain the following key-value pairs:

Key	Suggested Type	Value

message	String	"Command failed"
durationMS	Int32/Int64/Double	The execution time for the command in milliseconds. The calculated value MUST be the time to send the message and receive the reply from the server and MAY include BSON serialization and/or deserialization.
failure	Flexible	The error. The type and format of this value is flexible; see the logging specification for details on representing errors in log messages. If the command is considered sensitive, the error MUST be redacted and replaced with a language-appropriate alternative for a redacted error, e.g. an empty string, empty document, or null.

The unstructured form SHOULD be as follows, using the values defined in the structured format above to fill in placeholders as appropriate:

> Command "{{commandName}}" failed on database "{{databaseName}}" in {{durationMS}} ms using a connection with driver-generated ID {{driverConnectionId}} and server-generated ID {{serverConnectionId}} to {{serverHost}}:{{serverPort}} with service ID {{serviceId}}. The requestID is {{requestId}} and the operation ID is {{operationId}}. Error: {{error}}

Testing

See the README in the test directory for requirements and guidance.

Q&A

Why is the command document only available in the `CommandStartEvent`?

Some drivers may realize the command document as raw BSON, treating it as a component of the message transmitted to the server and stored in an internal buffer. By the time the server's response is received, this buffer may have been released. Requiring the retention of this buffer until command completion could result in unacceptable performance penalties, particularly when event listeners are introduced.

Changelog

- 2025-01-22: Clarify durationMS in logs may be Int32/Int64/Double.
- 2024-09-11: Migrated from reStructuredText to Markdown.
- 2015-09-16: Removed `limit` from find test with options to support 3.2.
  Changed find test read preference to `primaryPreferred`.

- 2015-10-01: Changed find test with a kill cursors to not run on server versions greater than 3.0. Added a find test with no kill cursors command which only runs on 3.1 and higher. Added notes on which tests should run based on server versions.
- 2015-10-19: Changed batchSize in the 3.2 find tests to expect the remaining value.
- 2015-10-31: Changed find test on 3.1 and higher to ignore being run on sharded clusters.
- 2015-11-22: Specify how to merge OP_MSG document sequences into command-started events.
- 2016-03-29: Added note on guarantee of the request ids.
- 2016-11-02: Added clause for not upconverting commands larger than maxBsonSize.
- 2018-04-16: Made inclusion of BSON serialization/deserialization in command durations to be optional.
- 2020-02-12: Added legacy hello `speculativeAuthenticate` to the list of values that should be redacted.
- 2021-04-15: Added `serviceId` field to events.
- 2021-05-05: Updated to use hello and legacy hello.
- 2021-08-30: Added `serverConnectionId` field to `CommandStartedEvent`, `CommandSucceededEvent` and `CommandFailedEvent`.
- 2022-05-18: Converted legacy tests to the unified test format.
- 2022-09-02: Remove material that only applies to MongoDB versions < 3.6.
- 2022-10-05: Remove spec front matter and reformat changelog.
- 2022-10-11: Add command logging information and tests.
- 2022-11-16: Update sensitive command tests to only run on server versions where the commands are supported.
- 2022-12-13: Updated log message `serverPort` field description to clarify drivers should populate it with the
  default port 27017 when relevant. Updated suggested unstructured forms of log messages to more clearly label connection IDs and use more readable server address representations.
- 2023-03-23: Updated `serverConnectionId` field to be Int64 as long-running servers can return Int64.
- 2023-06-13: Added `databaseName` field to `CommandFailedEvent` and `CommandSucceededEvent`.

Updated suggested unstructured forms of log messages reflecting the changes.

- 2023-10-19: Add Q&A section