8.0 Endpoints and Request Monitoring

The Request Monitoring feature enables you to configure logging of information related to requests, including metrics collected during request execution. This feature lets you enable logging of internal preset metrics for requests on specific endpoints. You can also log custom request data by calling the provided Request Logging APIs. This logged information may help you evaluate server performance.

This chapter provides an overview of creating endpoint declarations and using them to monitor requests on MarkLogic Server. This chapter includes the following sections:

8.1 Monitoring Requests

Using the Request Monitoring feature, you can switch on logging of internal preset metrics for requests on specific endpoints, or you can choose to log additional custom request data by calling the request logging APIs. The custom request data might contain a query plan, traces, or whatever information you want to collect and log for a request. This logged information may help you spot offending requests and evaluate request history.

8.2 App Server Request Monitoring

You can trigger request logging for an App Server through one or more of the following options:

- For targeted endpoints (main modules), by switching on monitoring in their endpoint declaration.
- For all requests on the App Server, by using a special server declaration.
- By calling request logging APIs in modules.

To switch on monitoring for an endpoint, you must add a `monitoring` section to the App Server Endpoint Declaration file and specify which metrics will be logged. Request monitoring is switched off by default for all metrics.

8.3 XDBC Server Request Monitoring

XDBC Server enables XCC and XDBC applications to communicate with MarkLogic Server. You can configure request monitoring for XDBC requests for specific endpoints and globally for the XDBC Server. There are two types of XDBC requests where request monitoring are enabled:

- XDBC Invoke Requests
- XDBC Eval Requests

8.3.1 XDBC Invoke Requests

You can enable request monitoring at both the endpoint level and at the server level for XDBC invoke request. For a specific endpoint, you must add a `monitoring` section to the XDBC Server Endpoint Declaration file and specify which metrics will be logged. To configure monitoring on a global level, you must add a `default.api` file in the modules root directory for the XDBC Server. For the Task Server, as there is no port number, the output request log file will be logged into a new type of log file called `TaskServer_RequestLog.txt`

8.3.2 XDBC Eval Requests

For XDBC eval requests, request monitoring is only available at the server level, as there are no real endpoints. To seperate the monitoring configuration between the invoke and eval request, and to add more control over the monitoring of the eval request, you can add an `eval.api` file to the module root in addition to the `default.api` file. The `eval.api` file has the same format as the `default.api` file, which contains only a monitoring section, but the settings in `eval.api` override those in `default.api`.

8.4 Task Server Monitoring

The Task Server processes request that has been spawned, such as from `xdmp:spawn()` or from a post-commit trigger action. Since each task is handled as a module, you can configure request monitoring for both endpoint level and server level. To configure request monitoring for a specific endpoint, add a `monitoring` section to the Task Server Endpoint Declaration file (`*.api`) and specify which metrics are to be logged. For monitoring on a global level, add a `monitoring` section to the `default.api` file in the modules root directory for the Task Server. The configuration in a specific endpoint declaration file overrides the settings in the global `default.api`.

8.5 Creating Endpoint Declarations

An Endpoint Declaration is a JSON file with the extension `.api` that resides in the **module** database or file directory of an HTTP server. The App Server uses the declarations in this file to dispatch requests to corresponding main modules. The declarations in this file also determine which requests are to be logged.

8.5.1 The Endpoint Declaration File

The name, parameters, and return value for each endpoint are declared in the `*.api` file. The `*.api` file contains a JSON data structure with the following properties:

Property	Declares
`functionName`	The name used to call the endpoint, which must match the name (without the `.api` extension) of the declaration file.
`desc`	Optional; plain text documentation for the endpoint.
`params`	Optional; an array specifying the parameters of the endpoint. This is omitted for endpoints with no parameters. Parameter objects have `name`, `desc`, `datatype`, `nullable`, and `multiple` properties.
`return`	Optional; an object specifying the endpoint return value. This is omitted for endpoints with no return value. The child object has `desc`, `datatype`, `nullable`, and `multiple` properties.
`errorDetail`	Optional; specifies a value from the following enumeration to control whether error responses include stack traces: • `log` (the default): log the stack trace on the server but do not return the stack trace to the middle tier. • `return`: include the stack trace in the exception on the middle tier as well as log it on the server.

Note: When monitoring a module that is not defined as an endpoint, none of the properties defined in the preceding table are needed

The following is a list of meters that can be logged with the parameters that control them:

Monitoring Flag	Data Type	Default Value	Parameters
`general`	`object`		Enables all the general (non-custom) meters. The list of parameters on which you can set constraints is in the next table.
`enabled`	`boolean`	`false`	Controls the logging of meters.
`custom`	`boolean`	`true`	Custom meters manipulated with the `xdmp:request-log-*` APIs.

Monitoring Flag	Data Type	Default Value	Parameters
`fragments`	`integer`	`0`	The maximum number of items to log. For each fragment: `root`, `expandedTreeCacheHits`, `expandedTreeCacheMisses`
`documents`	`integer`	`0`	The maximum number of items to log. For each document: `uri`, `expandedTreeCacheHits`, `expandedTreeCacheMisses`
`hosts`	`integer`	`0`	The maximum number of items to log. For each host: `host`, `roundTripTime`, `roundTripCount`

The following parameters may be included in a `*.api` file:

Parameter	Description
`commitTime`	The aggregate commit phase time, represented as a double-precision value in seconds.
`compileTime`	The aggregate time spent compiling a module or a program, represented as a double-precision value in seconds.
`compressedTreeSize`	The aggregate size in bytes read from disk by unsuccessful compressed tree cache lookups. Each unsuccessful compressed tree cache lookup is followed by a disk access to load the compressed tree into the cache.
`compressedTreeCacheHits`	The number of successful compressed tree cache lookups. The compressed tree cache holds XML document data in the compressed representation stored on disk.
`compressedTreeCacheMisses`	The number of unsuccessful compressed tree cache lookups. Each unsuccessful compressed tree cache lookup was followed by a disk access to load the compressed tree into the cache.
`contemporaneousTimestampTime`	The time spent by queries waiting for the contemporaneous timestamp for which any transaction is known to have committed, represented as a double-precision value in seconds. When the multi-version concurrency control is set contemporaneous, queries can block waiting for the contemporaneous transactions to fully commit.

Parameter	Description
`dbLibraryModuleCacheHits`	The number of library module cache hits from library modules from the modules database.
`dbLibraryModuleCacheMisses`	The number of library module cache misses from library modules from the modules database.
`dbMainModuleSequenceCacheHits`	The number of main module cache hits from main modules in a database.
`dbMainModuleSequenceCacheMisses`	The number of main module cache misses from main modules in a database.
`dbProgramCacheHits`	The number of module cache hits from the entire program made from modules in a database (may contain library modules from the special Modules directory).
`dbProgramCacheMisses`	The number of module cache misses from the entire program made from modules in a database (may contain library modules from the special Modules directory).
`elapsedTime`	The time elapsed since the start of the processing of this query, in the form of a duration. Use this parameter instead of the deprecated `xdmp:set-request-time-limit` function.
`envProgramCacheHits`	The number of module cache hits from the entire program made from ad hoc XSLT stylesheet nodes.
`envProgramCacheMisses`	The number of module cache misses from the entire program made from ad hoc XSLT stylesheet nodes.
`expandedTreeCacheHits`	The number of successful expanded tree cache lookups. The expanded tree cache cache holds XML document data in the expanded representation used by the XQuery evaluator.
`expandedTreeCacheMisses`	The number of unsuccessful expanded tree cache lookups. Each unsuccessful expanded tree lookup was followed by a compressed tree cache lookup to load the expanded tree into the cache.
`filterHits`	The number of successful search filter matches.
`filterMisses`	The number of unsuccessful search filter matches.
`fragmentsAdded`	The number of XML fragments added to the database by an update.

Parameter	Description
fragmentsDeleted	The number of XML fragments deleted from the database by an update.
fsLibraryModuleCacheHits	The number of library module cache hits from library modules on the file system.
fsLibraryModuleCacheMisses	The number of library module cache misses from library modules on the file system.
fsMainModuleSequenceCacheHits	The number of main module cache hits from main modules on the file system.
fsMainModuleSequenceCacheMisses	The number of main module cache misses from main modules on the file system.
fsProgramCacheHits	The number of module cache hits from the entire program made from modules on the file system.
fsProgramCacheMisses	The number of module cache misses from the entire program made from modules on the file system.
inMemoryCompressedTreeHits	The number of successful compressed tree lookups in in-memory stands.
inMemoryListHits	The number of successful list lookups in in-memory stands.
indexingTime	The indexing time of documents before they are inserted into the database, represented as a double-precision value in seconds.
linkCacheHits	The number of successful link cache lookups. The link cache is a transient cache that exists only for the duration of one query. It holds pointers to expanded trees, and is used to accelerate the frequent dereferencing of link nodes.
linkCacheMisses	The number of unsuccessful link cache lookups. Each unsuccessful link cache lookup was followed by a search for the link target tree.
listSize	The aggregate size in bytes read from disk by unsuccessful list cache lookups. Each unsuccessful list cache lookup is followed by a disk access to load the search term list into the cache.

Parameter	Description
listCacheHits	The number of successful list cache lookups. The list cache holds search termlists used to accelerate path expressions and text searches.
listCacheMisses	The number of unsuccessful list cache lookups. Each unsuccessful list cache lookup was followed by a disk access to load the search termlist into the cache.
lockTime	The aggregate time forests spend waiting for transactional read and write locks, represented as a double-precision value in seconds. This time can exceed the run-time.
readLocks	The number of read locks.
regexpCacheHits	The number of successful regular expression cache lookups. The regular expression cache is a transient cache that exists only for the duration of one query. It holds compiled regular expressions, and is used to accelerate the frequent use of regular expressions during the evaluation of a query.
regexpCacheMisses	The number of unsuccessful regular expression cache lookups. Each unsuccessful regular expression cache lookup was followed by a compilation of a regular expression from source text.
requests	The number of requests.
runTime	The aggregate time spent evaluating or running a module or a program, represented as a double-precision value in seconds.
valueCacheHits	The number of successful value cache lookups. The value cache is a transient cache that exists only for the duration of one query. It holds typed values, and is used to accelerate the frequent conversion of nodes to typed values.
valueCacheMisses	The number of unsuccessful value cache lookups. Each unsuccessful value cache lookup was followed by a conversion of an XML node to a typed value.
writeLocks	The number of write locks.

8.5.2 Constraints on Meters

To control the number of meters that are logged, you can put the following constraints on meters:

Operator	Description
`lt`	Less than
`gt`	Greater than
`le`	Less than or equal to
`ge`	Greater than or equal to

The declaration format of a constraint is:

```
meter_name : {"operator":value, "operator":value, ...}
```

For example:

```
"constraints": {
  "tripleCacheHits" : { "ge":1 }
}
```

In this example, `tripleCacheHits` is logged only if the the value of `tripleCacheHits` is >= 1.

Meters with zero or empty values are not normally logged. This is done to minimize the size of the Request Log file. To log a zero or empty value, use the following code:

```
"constraints": {
  "meter_name" : { "ge":0 }
}
```

The default constraint value on any meter is:

```
"gt":0
```

8.5.3 Controlling Request Logging Using Thresholds

You can add thresholds to specify that a request is logged only when the threshold conditions are satisfied. To declare a threshold, create a `thresholds` section in your `*.api` file as follows:

```
        {
          "monitoring":{
            "thresholds" : {
              "elapsedTime": { "gt": 1.0 }
            }
           "general:
             "enabled": true
          }
        }
```

The following operators are allowed in thresholds:

Operator	Description
`lt`	Less than
`gt`	Greater than
`le`	Less than or equal to
`ge`	Greater than or equal to

To disable threshold checks on custom meters so the meters can always be logged, set the boolean flag `excludeCustom` to true in the `thresholds` section, as follows:

```
        {
          "monitoring":{
            "thresholds" : {
              "excludeCustom": true,
              "elapsedTime": { "gt": 1.0 }
            }
           "general:
             "enabled": true
          }
        }
```

In the following example, the `general` meters for the request are logged only when the total runtime of the request is greater than one second:

```
{
  "monitoring":{
    "thresholds" : {
      "elapsedTime": { "gt": 1.0 }
    }
    "general" : {
      "enabled": true
    }
  }
}
```

8.5.4 Enabling Request Monitoring

Request monitoring is enabled by default on the following default MarkLogic application servers:

- The default App-Services application server, normally Port 8000:
  - For all requests related to the Query Console.
  - For requests with runtime exceeding one second that are not related to the Query Console. This mainly covers the REST Client API.
- The default Manage application server, normally Port 8002, for requests running longer than one second. This covers the Configuration Manager and Monitoring Dashboard.

If request monitoring is not already enabled, you can enable request monitoring by calling any server-side JavaScript (`*.sjs`) or XQuery (`*.xqy`) functions in files that reside in the **modules** directory as declared on the HTTP server via the Admin interface, and you can create other `*.api` files in the same directory or, if using the file system, in the same subdirectory as the JavaScript or XQuery file being called. For more information on configuring HTTP servers and the **modules** directory, see the "HTTP Servers" chapter in the *Administrator's Guide*.

The following example enables request monitoring:

```
{
"monitoring":{
  "general":{
    "enabled":true
  },
  "custom":true
}
}
```

The following example logs `tripleCacheHits` if the the value of `tripleCacheHits` is >= 0:

```
        "constraints": {
             "tripleCacheHits" : { "ge":0 }
            }
```

The following code fragment logs up to 10 documents, 10 fragments, and 4 hosts:

```
        "documents": 10,
        "fragments" : 10,
        "hosts" : 4
```

The following example is called `countdocs.api`. The function `getCount` is defined in a file called `countdocs.sjs` that resides in the same directory. The `general` object under the `monitoring` section of the file has `enable` and `custom` set to `true`; this enables request logging.

```
        {
          "functionName":"getCount",
          "params":[
            {
              "name":"collection",
              "datatype":"string",
              "multiple":false,
              "nullable":false
            },
            {
              "name":"method",
              "datatype":"int",
              "multiple":false,
              "nullable":false
            }

          ],
          "return":{
            "datatype":"string",
            "nullable":false,
            "multiple":false
          },

          "monitoring":{
            "general": {
              "enabled": true,
              "constraints": {
                "tripleCacheHits" : { "ge":1 }
              }
            },
            "custom": true ,
            "documents": 10,
            "fragments" : 0,
            "hosts" : 4
          }
        }
```

8.5.5 The Default Declaration File

You can configure request monitoring globally for all requests on an App Server by adding a `default.api` file where the **modules** root is configured for the App Server (either to the **modules** database or to the file system). The `default.api` is a JSON file that only contains a monitoring section.

The following is a sample `default.api` file:

```
{
"monitoring":{
    "general":  {
      "enabled": true,
      "constraints": {
        "tripleCacheHits" : { "ge":0 }
      }
    },
    "custom": true,
    "documents": 10,
    "fragments" : 10,
    "hosts" : 5  }
}
```

When you make a request to an endpoint and do not specify an App Server Endpoint Declaration file, the `default.api` file in the module database or file directory is used if it exists. If a module has a `*.api` file associated with it, the monitoring settings in the `*.api` file for that module are used instead of those in the `default.api` file.

If you want to save the `default.`api file, use JavaScript:

```
declareUpdate();
xdmp.documentInsert("/default.api",{
 "monitoring":{
  "general": {
   "enabled": true
   },
  "custom": true,
  "documents": 10,
  "fragments" : 10,
  "hosts" : 5 }
});
```

8.5.6 Request Logs

The logs for MarkLogic Server containing information about the requests you have chosen to log are stored in the `Logs` directory under the MarkLogic Server data directory (typically `c:\Program Files\MarkLogic\Data\Logs` on Windows, `/var/opt/MarkLogic/Logs` on UNIX-based systems) in the file *port_no*`_RequestLog.txt`. Each RequestLog.txt file will contain the meters from multiple monitored endpoints that:

- are configured on an App Server with some monitoring switched on.
- have calls to `xdmp:request-log-put` in their module.

The collected data is logged in JSON format at the rate of one line per request information. Even if monitoring is completely switched off on an endpoint, calls to `xdmp:request-log-put(`*key*`,`*value*`)` during a request will result in data being logged for all the `(`*key*`,`*value*`)` pairs that have been stored during the request.

The following is an example of a request log where different meters are logged for two endpoints:

```
{"time":"2018-11-09T14:07:14-08:00",
"url":"/booleanApiDecl.sjs?booleanArg=true", "user":"admin",
"result":10, "elapsedTime":0.007145, "requests":1,
"dbProgramCacheMisses":1, "dbMainModuleSequenceCacheMisses":1,
"dbLibraryModuleCacheMisses":0}

{"time":"2018-11-09T14:10:18-08:00",
"url":"/booleanApiDecl.xqy?booleanArg=true", "user":"admin",
"result":10, "elapsedTime":0.001603, "requests":1,
"dbProgramCacheMisses":1, "dbMainModuleSequenceCacheMisses":1,
"dbLibraryModuleCacheMisses":0}
```

8.6 Request Cancelling

This section describes the procedure to setup and enable request cancelling for an endpoint, a main module, or globally on an App Server or an XDBC Server. Request cancelling is disabled by default for all meters. You can enable request cancelling by adding a `limits` section to the monitoring section in the endpoint declaration, as in the following example:

```
{
  "monitoring": {
    "limits" : {
      "lockCount" : 100,
      "readSize" : 1000000
    }
  }
}
```

The following limits are available:

Meter	Unit	Description
`elapsedTime`	seconds	Equivalent to calling `xdmp:set-request-time-limit()`
`readSize`	bytes	Combined size read from disk (`listSize` + `compressedTreeSize`)
`lockCount`	count	Combined count for the number of times a read or a write lock was acquired (`readLocks` + `writeLocks`)

You can update the `limits` configuration while the server is running without having to restart the server.

8.7 Request Monitoring APIs

The Request Monitoring APIs are XQuery and JavaScript functions enable you to log additional information in request logs. The following are the request monitoring functions available:

- `xdmp:request-log-put`
- `xdmp:request-log-get`
- `xdmp:request-log-delete`
- `xdmp:set-request-limit`
- `xdmp.requestLogPut`
- `xdmp.requestLogGet`
- `xdmp.requestLogDelete`
- `xdmp:request-status`

Whatever is logged with `xdmp:request-log-put` displays in the log files unless the `custom` flag is set to `false`. When the `custom` flag is set to `false`, all custom logging is muted.

For more details about syntax and usage of these functions, see the AppServer functions in the *MarkLogic XQuery and XSLT Function Reference* and the *MarkLogic JavaScript Reference Guide*.