NDB Cluster Internals Manual

Abstract

This Manual contains information about the NDBCLUSTER storage engine that is not strictly necessary for running the NDB Cluster product, but can prove useful for development and debugging purposes. Topics covered in this Guide include, among others, communication protocols employed between nodes, file systems used by management nodes and data nodes, error messages, and debugging (DUMP) commands in the management client.

The information presented in this guide is current for recent releases of NDB Cluster up to and including NDB Cluster 7.5.5. Due to significant functional and other changes in NDB Cluster and its underlying APIs, you should not expect this information to apply to previous releases of the NDB Cluster software prior to NDB Cluster 7.2. Users of older NDB Cluster releases should upgrade to the latest available GA release of NDB Cluster 7.5.

For help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists, where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2016-12-15 (revision: 50182)

Table of Contents

1 NDB Cluster File Systems 1
1.1 NDB Cluster Data Node File System 1
1.1.1 NDB Cluster Data Node Data Directory Files 1
1.1.2 NDB Cluster Data Node File System Directory Files 2
1.1.3 NDB Cluster Data Node Backup Data Directory Files 3
1.1.4 NDB Cluster Disk Data Files 3
1.2 NDB Cluster Management Node File System 4
2 NDB Cluster Management Client DUMP Commands 5
2.1 DUMP 1 8
2.2 DUMP 13 10
2.3 DUMP 14 10
2.4 DUMP 15 11
2.5 DUMP 16 11
2.6 DUMP 17 11
2.7 DUMP 18 12
2.8 DUMP 20 12
2.9 DUMP 21 13
2.10 DUMP 22 13
2.11 DUMP 23 13
2.12 DUMP 24 15
2.13 DUMP 25 15
2.14 DUMP 70 15
2.15 DUMP 400 16
2.16 DUMP 401 16
2.17 DUMP 402 17
2.18 DUMP 403 17
2.19 DUMP 404 (OBSOLETE) 18
2.20 DUMP 908 18
2.21 DUMP 1000 19
2.22 DUMP 1223 19
2.23 DUMP 1224 20
2.24 DUMP 1225 20
2.25 DUMP 1226 20
2.26 DUMP 1228 20
2.27 DUMP 1229 21
2.28 DUMP 1332 21
2.29 DUMP 1333 22
2.30 DUMP 2300 23
2.31 DUMP 2301 23
2.32 DUMP 2302 24
2.33 DUMP 2303 25
2.34 DUMP 2304 25
2.35 DUMP 2305 28
2.36 DUMP 2308 28
2.37 DUMP 2315 28
2.38 DUMP 2350 29
2.39 DUMP 2352 30
2.40 DUMP 2354 31
2.41 DUMP 2398 31
2.42 DUMP 2399 32
2.43 DUMP 2400 33

2.44 DUMP 2401 33
2.45 DUMP 2402 34
2.46 DUMP 2403 34
2.47 DUMP 2404 35
2.48 DUMP 2405 35
2.49 DUMP 2406 36
2.50 DUMP 2500 36
2.51 DUMP 2501 37
2.52 DUMP 2502 37
2.53 DUMP 2503 (OBSOLETE) 38
2.54 DUMP 2504 39
2.55 DUMP 2505 40
2.56 DUMP 2506 (OBSOLETE) 40
2.57 DUMP 2507 41
2.58 DUMP 2508 41
2.59 DUMP 2509 41
2.60 DUMP 2510 41
2.61 DUMP 2511 42
2.62 DUMP 2512 42
2.63 DUMP 2513 42
2.64 DUMP 2514 43
2.65 DUMP 2515 43
2.66 DUMP 2516 44
2.67 DUMP 2517 44
2.68 DUMP 2550 45
2.69 DUMP 2555 45
2.70 DUMP 2600 45
2.71 DUMP 2601 46
2.72 DUMP 2602 46
2.73 DUMP 2603 46
2.74 DUMP 2604 47
2.75 DUMP 5900 47
2.76 DUMP 7000 48
2.77 DUMP 7001 48
2.78 DUMP 7002 48
2.79 DUMP 7003 49
2.80 DUMP 7004 49
2.81 DUMP 7005 49
2.82 DUMP 7006 49
2.83 DUMP 7007 50
2.84 DUMP 7008 50
2.85 DUMP 7009 50
2.86 DUMP 7010 51
2.87 DUMP 7011 51
2.88 DUMP 7012 51
2.89 DUMP 7013 52
2.90 DUMP 7014 52
2.91 DUMP 7015 52
2.92 DUMP 7016 53
2.93 DUMP 7017 53
2.94 DUMP 7018 54
2.95 DUMP 7019 54
2.96 DUMP 7020 54
2.97 DUMP 7021 55

2.98 DUMP 7024 56
2.99 DUMP 7033 56
2.100 DUMP 7080 57
2.101 DUMP 7090 57
2.102 DUMP 7098 57
2.103 DUMP 7099 57
2.104 DUMP 7901 58
2.105 DUMP 8004 58
2.106 DUMP 8005 58
2.107 DUMP 8010 59
2.108 DUMP 8011 59
2.109 DUMP 8013 60
2.110 DUMP 9002 60
2.111 DUMP 9800 60
2.112 DUMP 9801 61
2.113 DUMP 9802 61
2.114 DUMP 9803 62
2.115 DUMP 10000 62
2.116 DUMP 11000 62
2.117 DUMP 12001 62
2.118 DUMP 12002 63
2.119 DUMP 12009 63
3 The NDB Communication Protocol 65
3.1 NDB Protocol Overview 65
3.2 NDB Protocol Messages 66
3.3 Operations and Signals 66
4 NDB Kernel Blocks 77
4.1 The BACKUP Block 78
4.2 The CMVMI Block 78
4.3 The DBACC Block 78
4.4 The DBDICT Block 79
4.5 The DBDIH Block 79
4.6 The DBINFO Block 80
4.7 The DBLQH Block 80
4.8 The DBSPJ Block 82
4.9 The DBTC Block 82
4.10 The DBTUP Block 84
4.11 The DBTUX Block 85
4.12 The DBUTIL Block 86
4.13 The LGMAN Block 86
4.14 The NDBCNTR Block 86
4.15 The NDBFS Block 86
4.16 The PGMAN Block 87
4.17 The QMGR Block 87
4.18 The RESTORE Block 88
4.19 The SUMA Block 88
4.20 The THRMAN Block 88
4.21 The TRPMAN Block 89
4.22 The TSMAN Block 89
4.23 The TRIX Block 89
5 NDB Cluster Start Phases 91
5.1 Initialization Phase (Phase -1) 91
5.2 Configuration Read Phase (STTOR Phase -1) 92
5.3 STTOR Phase 0 93

5.4 STTOR Phase 1 94
5.5 STTOR Phase 2 97
5.6 NDB_STTOR Phase 1 97
5.7 STTOR Phase 3 97
5.8 NDB_STTOR Phase 2 97
5.9 STTOR Phase 4 98
5.10 NDB_STTOR Phase 3 98
5.11 STTOR Phase 5 99
5.12 NDB_STTOR Phase 4 99
5.13 NDB_STTOR Phase 5 99
5.14 NDB_STTOR Phase 6 100
5.15 STTOR Phase 6 100
5.16 STTOR Phase 7 101
5.17 STTOR Phase 8 101
5.18 NDB_STTOR Phase 7 101
5.19 STTOR Phase 9 101
5.20 STTOR Phase 101 101
5.21 System Restart Handling in Phase 4 101
5.22 START_MEREQ Handling 102
6 NDB Schema Object Versions 103
7 NDB Cluster API Errors 107
7.1 Data Node Error Messages 107
7.1.1 ndbd Error Codes 107
7.1.2 ndbd Error Classifications 111
7.2 NDB Transporter Errors 112
A NDB Internals Glossary 115
Index 117

Chapter 1 NDB Cluster File Systems

Table of Contents

1.1 NDB Cluster Data Node File System 1
1.1.1 NDB Cluster Data Node Data Directory Files 1
1.1.2 NDB Cluster Data Node File System Directory Files 2
1.1.3 NDB Cluster Data Node Backup Data Directory Files 3
1.1.4 NDB Cluster Disk Data Files 3
1.2 NDB Cluster Management Node File System 4

This chapter contains information about the file systems created and used by NDB Cluster data nodes and management nodes.

1.1 NDB Cluster Data Node File System

This section discusses the files and directories created by NDB Cluster nodes, their usual locations, and their purpose.

1.1.1 NDB Cluster Data Node Data Directory Files

An NDB Cluster data node's data directory (`DataDir`) contains at least 3 files. These are named as shown in the following list, where *`node_id`* is the node ID:

- `ndb_node_id_out.log`

  Sample output:

```
2015-11-01 20:13:24 [ndbd] INFO     -- Angel pid: 13677 ndb pid: 13678
2015-11-01 20:13:24 [ndbd] INFO     -- NDB Cluster -- DB node 1
2015-11-01 20:13:24 [ndbd] INFO     -- Version 5.6.27-ndb-7.4.8 --
2015-11-01 20:13:24 [ndbd] INFO     -- Configuration fetched at localhost port 1186
2015-11-01 20:13:24 [ndbd] INFO     -- Start initiated (version 5.6.27-ndb-7.4.8)
2015-11-01 20:13:24 [ndbd] INFO     -- Ndbd_mem_manager::init(1) min: 20Mb initial: 20Mb
WOPool::init(61, 9)
RWPool::init(82, 13)
RWPool::init(a2, 18)
RWPool::init(c2, 13)
RWPool::init(122, 17)
RWPool::init(142, 15)
WOPool::init(41, 8)
RWPool::init(e2, 12)
RWPool::init(102, 55)
WOPool::init(21, 8)
Dbdict: name=sys/def/SYSTAB_0,id=0,obj_ptr_i=0
Dbdict: name=sys/def/NDB$EVENTS_0,id=1,obj_ptr_i=1
m_active_buckets.set(0)
```

- `ndb_node_id_signal.log`

  This file contains a log of all signals sent to or from the data node.

Note

This file is created only if the `SendSignalId` parameter is enabled, which is true only for `-debug` builds.

- `ndb_node_id.pid`

  This file contains the data node's process ID; it is created when the `ndbd` process is started.

The location of these files is determined by the value of the `DataDir` configuration parameter.

1.1.2 NDB Cluster Data Node File System Directory Files

The location of this directory can be set using `FileSystemPath`; the directory itself is always named `ndb_nodeid_fs`, where *`nodeid`* is the data node's node ID. The file system directory contains the following files and directories:

Files:

- `data-nodeid.dat`
- `undo-nodeid.dat`

Directories:

- `LCP`: This directory holds 2 subdirectories, named `0` and `1`, each of which which contain local checkpoint data files, one per local checkpoint.

  These subdirectories each contain a number of files whose names follow the pattern `TNFM.Data`, where *`N`* is a table ID and M is a fragment number. Each data node typically has one primary fragment and one backup fragment. This means that, for an NDB Cluster having 2 data nodes, and with `NoOfReplicas` equal to 2, *`M`* is either 0 to 1. For a 4-node cluster with `NoOfReplicas` equal to 2, *`M`* is either 0 or 2 on node group 1, and either 1 or 3 on node group 2.

  When using `ndbmtd` there may be more than one primary fragment per node. In this case, *`M`* is a number in the range of 0 to the number of LQH worker threads in the entire cluster, less 1. The number of fragments on each data node is equal to the number of LQH on that node times `NoOfReplicas`.

  Note

  Increasing `MaxNoOfExecutionThreads` does not change the number of fragments used by existing tables; only newly-created tables automatically use the new fragment count. To force the new fragment count to be used by an existing table after increasing `MaxNoOfExecutionThreads`, you must perform an `ALTER TABLE ... REORGANIZE PARTITION` statement (just as when adding new node groups).

- Directories named `D1` and `D2`, each of which contains 2 subdirectories:
  - `DBDICT`: Contains data dictionary information. This is stored in:
    - The file `P0.SchemaLog`
    - A set of directories `T0`, `T1`, `T2`, ..., each of which contains an `S0.TableList` file.
  - Directories named `D8`, `D9`, `D10`, and `D11`, each of which contains a directory named `DBLQH`. These contain the redo log, which is divided into four parts that are stored in these directories. with redo log part 0 being stored in `D8`, part 1 in `D9`, and so on.

    Within each directory can be found a `DBLQH` subdirectory containing the *`N`* redo log files; these are named `S0.Fraglog`, `S1.FragLog`, `S2.FragLog`, ..., `SN.FragLog`, where *`N`* is equal

to the value of the `NoOfFragmentLogFiles` configuration parameter. The default value for `NoOfFragmentLogFiles` is 16. The default size of each of these files is 16 MB, controlled by the `FragmentLogFileSize` configuration parameter.

The size of each of the four redo log parts is `NoOfFragmentLogFiles * FragmentLogFileSize`. You can find out how much space the redo log is using with `DUMP 2398` or `DUMP 2399`; see Section 2.41, “DUMP 2398”, and Section 2.42, “DUMP 2399”, for more information.

- `DBDIH`: This directory contains the file `P`*`X`*`.sysfile`, which records information such as the last GCI, restart status, and node group membership of each node; its structure is defined in `storage/ndb/src/kernel/blocks/dbdih/Sysfile.hpp` in the NDB Cluster source tree. In addition, the `S`*`X`*`.FragList` files keep records of the fragments belonging to each table.

1.1.3 NDB Cluster Data Node Backup Data Directory Files

NDB Cluster creates backup files in the directory specified by the `BackupDataDir` configuration parameter, as discussed in Using The NDB Cluster Management Client to Create a Backup.

See NDB Cluster Backup Concepts, for information about the files created when a backup is performed.

1.1.4 NDB Cluster Disk Data Files

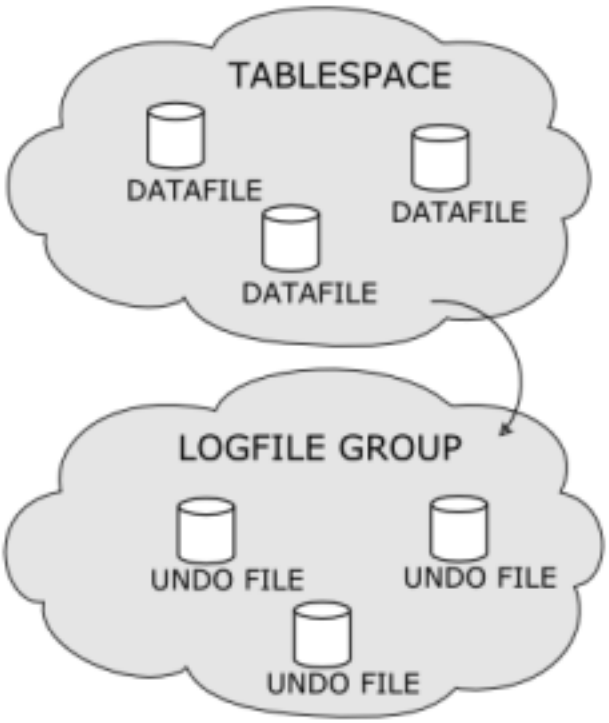
Note

This section applies only to NDB Cluster in MySQL 5.1 and later. Previous versions of MySQL did not support Disk Data tables.

NDB Cluster Disk Data files are created (or dropped) by the user by means of SQL statements intended specifically for this purpose. Such files include the following:

- One or more *undo logfiles* associated with a *logfile group*
- One or more *datafiles* associated with a *tablespace* that uses the logfile group for undo logging

Both undo logfiles and datafiles are created in the data directory (`DataDir`) of each cluster data node. The relationship of these files with their logfile group and tablespace are shown in the following diagram:



Disk Data files and the SQL commands used to create and drop them are discussed in depth in NDB Cluster Disk Data Tables.

1.2 NDB Cluster Management Node File System

The files used by an NDB Cluster management node are discussed in `ndb_mgmd` — The NDB Cluster Management Server Daemon.

Chapter 2 NDB Cluster Management Client DUMP Commands

Table of Contents

2.1 DUMP 1 8
2.2 DUMP 13 10
2.3 DUMP 14 10
2.4 DUMP 15 11
2.5 DUMP 16 11
2.6 DUMP 17 11
2.7 DUMP 18 12
2.8 DUMP 20 12
2.9 DUMP 21 13
2.10 DUMP 22 13
2.11 DUMP 23 13
2.12 DUMP 24 15
2.13 DUMP 25 15
2.14 DUMP 70 15
2.15 DUMP 400 16
2.16 DUMP 401 16
2.17 DUMP 402 17
2.18 DUMP 403 17
2.19 DUMP 404 (OBSOLETE) 18
2.20 DUMP 908 18
2.21 DUMP 1000 19
2.22 DUMP 1223 19
2.23 DUMP 1224 20
2.24 DUMP 1225 20
2.25 DUMP 1226 20
2.26 DUMP 1228 20
2.27 DUMP 1229 21
2.28 DUMP 1332 21
2.29 DUMP 1333 22
2.30 DUMP 2300 23
2.31 DUMP 2301 23
2.32 DUMP 2302 24
2.33 DUMP 2303 25
2.34 DUMP 2304 25
2.35 DUMP 2305 28
2.36 DUMP 2308 28
2.37 DUMP 2315 28
2.38 DUMP 2350 29
2.39 DUMP 2352 30
2.40 DUMP 2354 31
2.41 DUMP 2398 31
2.42 DUMP 2399 32
2.43 DUMP 2400 33
2.44 DUMP 2401 33
2.45 DUMP 2402 34
2.46 DUMP 2403 34
2.47 DUMP 2404 35
2.48 DUMP 2405 35

2.49 DUMP 2406 36
2.50 DUMP 2500 36
2.51 DUMP 2501 37
2.52 DUMP 2502 37
2.53 DUMP 2503 (OBSOLETE) 38
2.54 DUMP 2504 39
2.55 DUMP 2505 40
2.56 DUMP 2506 (OBSOLETE) 40
2.57 DUMP 2507 41
2.58 DUMP 2508 41
2.59 DUMP 2509 41
2.60 DUMP 2510 41
2.61 DUMP 2511 42
2.62 DUMP 2512 42
2.63 DUMP 2513 42
2.64 DUMP 2514 43
2.65 DUMP 2515 43
2.66 DUMP 2516 44
2.67 DUMP 2517 44
2.68 DUMP 2550 45
2.69 DUMP 2555 45
2.70 DUMP 2600 45
2.71 DUMP 2601 46
2.72 DUMP 2602 46
2.73 DUMP 2603 46
2.74 DUMP 2604 47
2.75 DUMP 5900 47
2.76 DUMP 7000 48
2.77 DUMP 7001 48
2.78 DUMP 7002 48
2.79 DUMP 7003 49
2.80 DUMP 7004 49
2.81 DUMP 7005 49
2.82 DUMP 7006 49
2.83 DUMP 7007 50
2.84 DUMP 7008 50
2.85 DUMP 7009 50
2.86 DUMP 7010 51
2.87 DUMP 7011 51
2.88 DUMP 7012 51
2.89 DUMP 7013 52
2.90 DUMP 7014 52
2.91 DUMP 7015 52
2.92 DUMP 7016 53
2.93 DUMP 7017 53
2.94 DUMP 7018 54
2.95 DUMP 7019 54
2.96 DUMP 7020 54
2.97 DUMP 7021 55
2.98 DUMP 7024 56
2.99 DUMP 7033 56
2.100 DUMP 7080 57
2.101 DUMP 7090 57
2.102 DUMP 7098 57

2.103 DUMP 7099 57
2.104 DUMP 7901 58
2.105 DUMP 8004 58
2.106 DUMP 8005 58
2.107 DUMP 8010 59
2.108 DUMP 8011 59
2.109 DUMP 8013 60
2.110 DUMP 9002 60
2.111 DUMP 9800 60
2.112 DUMP 9801 61
2.113 DUMP 9802 61
2.114 DUMP 9803 62
2.115 DUMP 10000 62
2.116 DUMP 11000 62
2.117 DUMP 12001 62
2.118 DUMP 12002 63
2.119 DUMP 12009 63

Warning

Never use these commands on a production NDB Cluster except under the express direction of MySQL Technical Support. Oracle will *not* be held responsible for adverse results arising from their use under any other circumstances!

`DUMP` commands can be used in the Cluster management client (`ndb_mgm`) to dump debugging information to the Cluster log. They are documented here, rather than in the *MySQL Manual*, for the following reasons:

- They are intended only for use in troubleshooting, debugging, and similar activities by MySQL developers, QA, and support personnel.
- Due to the way in which `DUMP` commands interact with memory, they can cause a running NDB Cluster to malfunction or even to fail completely when used.
- The formats, arguments, and even availability of these commands are not guaranteed to be stable. *All of this information is subject to change at any time without prior notice*.
- For the preceding reasons, `DUMP` commands are neither intended nor warranted for use in a production environment by end-users.

General syntax:

```
ndb_mgm> node_id DUMP code [arguments]
```

This causes the contents of one or more `NDB` registers on the node with ID *`node_id`* to be dumped to the Cluster log. The registers affected are determined by the value of *`code`*. Some (but not all) `DUMP` commands accept additional *`arguments`*; these are noted and described where applicable.

Individual `DUMP` commands are listed by their *`code`* values in the sections that follow. For convenience in locating a given `DUMP` code, they are divided by thousands.

Each listing includes the following information:

- The *`code`* value
- The relevant `NDB` kernel block or blocks (see Chapter 4, *NDB Kernel Blocks*, for information about these)
- The `DUMP` code symbol where defined; if undefined, this is indicated using a triple dash: `---`.

- Sample output; unless otherwise stated, it is assumed that each DUMP command is invoked as shown here:

```
ndb_mgm> 2 DUMP code
```

  Generally, this is from the cluster log; in some cases, where the output may be generated in the node log instead, this is indicated. Where the DUMP command produces errors, the output is generally taken from the error log.

- Where applicable, additional information such as possible extra *arguments*, warnings, state or other values returned in the DUMP command's output, and so on. Otherwise its absence is indicated with "[N/A]".

Note

DUMP command codes are not necessarily defined sequentially. For example, codes 2 through 12 are currently undefined, and so are not listed. However, individual DUMP code values are subject to change, and there is no guarantee that a given code value will continue to be defined for the same purpose (or defined at all, or undefined) over time.

There is also no guarantee that a given DUMP code—even if currently undefined—will not have serious consequences when used on a running NDB Cluster.

For information concerning other ndb_mgm client commands, see Commands in the NDB Cluster Management Client.

Note

DUMP codes in the following ranges are currently unused and thus unsupported:

- 3000 to 5000
- 6000 to 7000
- 13000 and higher

2.1 DUMP 1

Code	Symbol	Kernel Block(s)
1	[none]	QMGR

Description. Dumps information about cluster start Phase 1 variables (see Section 5.4, "STTOR Phase 1").

Sample Output.

```
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: creadyDistCom = 1, cpresident = 5

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: cpresidentAlive = 1, cpresidentCand = 5 (gci: 254325)

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: ctoStatus = 0

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 1: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 2: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 3: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 4: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 5: ZRUNNING(3)
```

```
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 6: ZRUNNING(3)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 7: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 8: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 9: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 10: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 11: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 12: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 13: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 14: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 15: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 16: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 17: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 18: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 19: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 20: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 21: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 22: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 23: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 24: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 25: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 26: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 27: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 28: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 29: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 30: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 31: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 32: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 33: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 34: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 35: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 36: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 37: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 38: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 39: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 40: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 41: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 42: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 43: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 44: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 45: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 46: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 47: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 5: Node 48: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: creadyDistCom = 1, cpresident = 5

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: cpresidentAlive = 0, cpresidentCand = 5 (gci: 254325)

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: ctoStatus = 0

2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 1: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 2: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 3: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 4: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 5: ZRUNNING(3)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 6: ZRUNNING(3)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 7: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 8: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 9: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 10: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 11: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 12: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 13: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 14: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 15: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 16: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 17: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 18: ZAPI_INACTIVE(7)
```

```
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 19: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 20: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 21: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 22: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 23: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 24: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 25: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 26: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 27: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 28: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 29: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 30: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 31: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 32: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 33: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 34: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 35: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 36: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 37: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 38: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 39: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 40: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 41: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 42: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 43: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 44: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 45: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 46: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 47: ZAPI_INACTIVE(7)
2014-10-13 20:54:29 [MgmtSrvr] INFO     -- Node 6: Node 48: ZAPI_INACTIVE(7)
```

Additional Information. [N/A]

2.2 DUMP 13

Code	Symbol	Kernel Block(s)
13	[none]	CMVMI, NDBCNTR

Description. Dump signal counter and start phase information.

Sample Output.

```
2014-10-13 20:56:33 [MgmtSrvr] INFO     -- Node 5: Cntr: cstartPhase = 9, cinternalStartphase = 8, block = 0
2014-10-13 20:56:33 [MgmtSrvr] INFO     -- Node 5: Cntr: cmasterNodeId = 5
2014-10-13 20:56:33 [MgmtSrvr] INFO     -- Node 6: Cntr: cstartPhase = 9, cinternalStartphase = 8, block = 0
2014-10-13 20:56:33 [MgmtSrvr] INFO     -- Node 6: Cntr: cmasterNodeId = 5
```

Additional Information. [N/A]

2.3 DUMP 14

Code	Symbol	Kernel Block(s)
14	CommitAckMarkersSize	DBLQH, DBTC

Description. Dumps free size in commitAckMarkerPool.

Sample Output.

```
2014-10-13 20:58:11 [MgmtSrvr] INFO     -- Node 5: LQH: m_commitAckMarkerPool: 36094 free size: 36094
```

```
2014-10-13 20:58:11 [MgmtSrvr] INFO     -- Node 6: LQH: m_commitAckMarkerPool: 36094 free size: 36094
```

Additional Information. [N/A]

2.4 DUMP 15

Code	Symbol	Kernel Block(s)
15	CommitAckMarkersDump	DBLQH, DBTC

Description. Dumps information in commitAckMarkerPool.

Sample Output.

```
2014-10-13 20:58:11 [MgmtSrvr] INFO     -- Node 5: LQH: m_commitAckMarkerPool: 36094 free size: 36094
2014-10-13 20:58:11 [MgmtSrvr] INFO     -- Node 6: LQH: m_commitAckMarkerPool: 36094 free size: 36094
```

Additional Information. [N/A]

2.5 DUMP 16

Code	Symbol	Kernel Block(s)
16	DihDumpNodeRestartInfo	DBDIH

Description. Provides node restart information.

Sample Output.

```
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 5: c_nodeStartMaster.blockGcp = 0, c_nodeStartMaster.wait =
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 5: [ 0 : cfirstVerifyQueue = 0 clastVerifyQueue = 0 sz: 819
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 5: cgcpOrderBlocked = 0
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 6: c_nodeStartMaster.blockGcp = 0, c_nodeStartMaster.wait =
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 6: [ 0 : cfirstVerifyQueue = 0 clastVerifyQueue = 0 sz: 819
2014-10-13 21:01:19 [MgmtSrvr] INFO     -- Node 6: cgcpOrderBlocked = 0
```

Additional Information. [N/A]

2.6 DUMP 17

Code	Symbol	Kernel Block(s)
17	DihDumpNodeStatusInfo	DBDIH

Description. Dumps node status.

Sample Output.

```
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 5: Printing nodeStatus of all nodes
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 5: Node = 5 has status = 1
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 5: Node = 6 has status = 1
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 6: Printing nodeStatus of all nodes
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 6: Node = 5 has status = 1
2014-10-13 21:02:28 [MgmtSrvr] INFO     -- Node 6: Node = 6 has status = 1
```

Additional Information. Possible node status values are shown in the following table:

Value	Name
0	NOT_IN_CLUSTER
1	ALIVE
2	STARTING
3	DIED_NOW
4	DYING
5	DEAD

2.7 DUMP 18

Code	Symbol	Kernel Block(s)
18	DihPrintFragmentation	DBDIH

Description. Prints one entry per table fragment; lists the table number, fragment number, log part ID, and the IDs of the nodes handling the primary and secondary replicas of this fragment.

Sample Output.

```
Node 5: Printing nodegroups --
Node 5: NG 0(0) ref: 4 [ cnt: 2 : 5 6 4294967040 4294967040 ]
Node 5: Printing fragmentation of all tables --
Node 5:  Table 2 Fragment 0(1) LP: 0 - 5 6
Node 5:  Table 2 Fragment 1(1) LP: 0 - 6 5
Node 5:  Table 3 Fragment 0(2) LP: 1 - 5 6
Node 5:  Table 3 Fragment 1(2) LP: 1 - 6 5
Node 6: Printing nodegroups --
Node 6: NG 0(0) ref: 4 [ cnt: 2 : 5 6 4294967040 4294967040 ]
Node 6: Printing fragmentation of all tables --
Node 6:  Table 2 Fragment 0(1) LP: 0 - 5 6
Node 6:  Table 2 Fragment 1(1) LP: 0 - 6 5
Node 6:  Table 3 Fragment 0(2) LP: 1 - 5 6
Node 6:  Table 3 Fragment 1(2) LP: 1 - 6 5
```

Additional Information. [N/A]

2.8 DUMP 20

Code	Symbol	Kernel Block(s)
20	[none]	BACKUP

Description. Prints the values of BackupDataBufferSize, BackupLogBufferSize, BackupWriteSize, and BackupMaxWriteSize

Sample Output.

```
2014-10-13 21:04:13 [MgmtSrvr] INFO     -- Node 5: Backup: data: 17039872 log: 17039872 min: 262144 max: 10485
2014-10-13 21:04:13 [MgmtSrvr] INFO     -- Node 6: Backup: data: 17039872 log: 17039872 min: 262144 max: 10485
```

Additional Information. This command can also be used to set these parameters, as in this example:

```
ndb_mgm> ALL DUMP 20 3 3 64 512
ALL DUMP 20 3 3 64 512
```

```
Sending dump signal with data:
0x00000014 0x00000003 0x00000003 0x00000040
0x00000200
Sending dump signal with data:
0x00000014 0x00000003 0x00000003 0x00000040
0x00000200

...

2014-10-13 21:05:52 [MgmtSrvr] INFO     -- Node 5: Backup: data: 3145728 log: 3145728 min: 65536 max: 5242
2014-10-13 21:05:52 [MgmtSrvr] INFO     -- Node 6: Backup: data: 3145728 log: 3145728 min: 65536 max: 5242
```

Warning

You must set each of these parameters to the same value on all nodes; otherwise, subsequent issuing of a `START BACKUP` command crashes the cluster.

2.9 DUMP 21

Code	Symbol	Kernel Block(s)
21	[none]	BACKUP

Description. Sends a `GSN_BACKUP_REQ` signal to the node, causing that node to initiate a backup.

Sample Output.

```
Node 2: Backup 1 started from node 2
Node 2: Backup 1 started from node 2 completed
StartGCP: 158515 StopGCP: 158518
#Records: 2061 #LogRecords: 0
Data: 35664 bytes Log: 0 bytes
```

Additional Information. [N/A]

2.10 DUMP 22

Code	Symbol	Kernel Block(s)
22 *backup_id*	[none]	BACKUP

Description. Sends a `GSN_FSREMOVEREQ` signal to the node. This should remove the backup having backup ID *backup_id* from the backup directory; *however, it actually causes the node to crash*.

Sample Output.

```
...
```

Additional Information. It appears that *any* invocation of `DUMP 22` causes the node or nodes to crash.

2.11 DUMP 23

Code	Symbol	Kernel Block(s)
23	[none]	BACKUP

Description. Dumps all backup records and file entries belonging to those records.

Note

The example shows a single record with a single file only, but there may be multiple records and multiple file lines within each record.

Sample Output. With no backup in progress (`BackupRecord` shows as `0`):

```
Node 2: BackupRecord 0: BackupId: 5 MasterRef: f70002 ClientRef: 0
Node 2:  State: 2
Node 2:  file 0: type: 3 flags: H'0
```

While a backup is in progress (`BackupRecord` is `1`):

```
Node 2: BackupRecord 1: BackupId: 8 MasterRef: f40002 ClientRef: 80010001
Node 2:  State: 1
Node 2:  file 3: type: 3 flags: H'1
Node 2:  file 2: type: 2 flags: H'1
Node 2:  file 0: type: 1 flags: H'9
Node 2: BackupRecord 0: BackupId: 110 MasterRef: f70002 ClientRef: 0
Node 2:  State: 2
Node 2:  file 0: type: 3 flags: H'0
```

Additional Information. Possible `State` values are shown in the following table:

Value	State	Description
0	`INITIAL`	
1	`DEFINING`	Defining backup content and parameters
2	`DEFINED`	`DEFINE_BACKUP_CONF` signal sent by slave, received on master
3	`STARTED`	Creating triggers
4	`SCANNING`	Scanning fragments
5	`STOPPING`	Closing files
6	`CLEANING`	Freeing resources
7	`ABORTING`	Aborting backup

Types are shown in the following table:

Value	Name
1	`CTL_FILE`
2	`LOG_FILE`
3	`DATA_FILE`
4	`LCP_FILE`

Flags are shown in the following table:

Value	Name
`0x01`	`BF_OPEN`
`0x02`	`BF_OPENING`
`0x04`	`BF_CLOSING`

Value	Name
0x08	BF_FILE_THREAD
0x10	BF_SCAN_THREAD
0x20	BF_LCP_META

2.12 DUMP 24

Code	Symbol	Kernel Block(s)
24	[none]	BACKUP

Description. Prints backup record pool information.

Sample Output.

```
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 5: Backup - dump pool sizes
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 5: BackupPool: 2 BackupFilePool: 4 TablePool: 323
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 5: AttrPool: 2 TriggerPool: 4 FragmentPool: 323
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 5: PagePool: 1579
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 6: Backup - dump pool sizes
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 6: BackupPool: 2 BackupFilePool: 4 TablePool: 323
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 6: AttrPool: 2 TriggerPool: 4 FragmentPool: 323
2014-10-13 21:11:31 [MgmtSrvr] INFO     -- Node 6: PagePool: 1579
```

Additional Information. If 2424 is passed as an argument (for example, 2 DUMP 24 2424), this causes an LCP.

2.13 DUMP 25

Code	Symbol	Kernel Block(s)
25	NdbcntrTestStopOnError	NDBCNTR

Description. Kills the data node or nodes.

Sample Output.

```
...
```

Additional Information. [N/A]

2.14 DUMP 70

Code	Symbol	Kernel Block(s)
70	NdbcntrStopNodes	NDBCNTR

Description. Forces data node shutdown.

Sample Output.

```
...
```

Additional Information. [N/A]

2.15 DUMP 400

Code	Symbol	Kernel Block(s)
400	NdbfsDumpFileStat-	NDBFS

Description. Provides NDB file system statistics.

Sample Output.

```
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 5: NDBFS: Files: 28 Open files: 10
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 5:  Idle files: 18 Max opened files: 12
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 5:  Bound Threads: 28 (active 10) Unbound threads: 2
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 5:  Max files: 0
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 5:  Requests: 256
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 6: NDBFS: Files: 28 Open files: 10
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 6:  Idle files: 18 Max opened files: 12
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 6:  Bound Threads: 28 (active 10) Unbound threads: 2
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 6:  Max files: 0
2014-10-13 21:16:26 [MgmtSrvr] INFO     -- Node 6:  Requests: 256
```

Additional Information. [N/A]

2.16 DUMP 401

Code	Symbol	Kernel Block(s)
401	NdbfsDumpAllFiles	NDBFS

Description. Prints NDB file system file handles and states (OPEN or CLOSED).

Sample Output.

```
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: NDBFS: Dump all files: 28
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  0 (0x7f5aec0029f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  1 (0x7f5aec0100f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  2 (0x7f5aec01d780): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  3 (0x7f5aec02add0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  4 (0x7f5aec0387f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  5 (0x7f5aec045e40): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  6 (0x7f5aec053490): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  7 (0x7f5aec060ae0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  8 (0x7f5aec06e130): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5:  9 (0x7f5aec07b780): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 10 (0x7f5aec088dd0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 11 (0x7f5aec0969f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 12 (0x7f5aec0a4040): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 13 (0x7f5aec0b1690): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 14 (0x7f5aec0bece0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 15 (0x7f5aec0cc330): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 16 (0x7f5aec0d9980): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 17 (0x7f5aec0e6fd0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 18 (0x7f5aec0f4620): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 19 (0x7f5aec101c70): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 20 (0x7f5aec10f2c0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 21 (0x7f5aec11c910): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 22 (0x7f5aec129f60): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 23 (0x7f5aec1375b0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 24 (0x7f5aec144c00): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 25 (0x7f5aec152250): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 26 (0x7f5aec15f8a0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 5: 27 (0x7f5aec16cef0): OPEN
```

```
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: NDBFS: Dump all files: 28
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  0 (0x7fa0300029f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  1 (0x7fa0300100f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  2 (0x7fa03001d780): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  3 (0x7fa03002add0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  4 (0x7fa0300387f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  5 (0x7fa030045e40): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  6 (0x7fa030053490): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  7 (0x7fa030060ae0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  8 (0x7fa03006e130): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6:  9 (0x7fa03007b780): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 10 (0x7fa030088dd0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 11 (0x7fa0300969f0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 12 (0x7fa0300a4040): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 13 (0x7fa0300b1690): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 14 (0x7fa0300bece0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 15 (0x7fa0300cc330): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 16 (0x7fa0300d9980): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 17 (0x7fa0300e6fd0): CLOSED
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 18 (0x7fa0300f4620): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 19 (0x7fa030101c70): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 20 (0x7fa03010f2c0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 21 (0x7fa03011c910): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 22 (0x7fa030129f60): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 23 (0x7fa0301375b0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 24 (0x7fa030144c00): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 25 (0x7fa030152250): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 26 (0x7fa03015f8a0): OPEN
2014-10-13 21:17:37 [MgmtSrvr] INFO     -- Node 6: 27 (0x7fa03016cef0): OPEN
```

Additional Information. [N/A]

2.17 DUMP 402

Code	Symbol	Kernel Block(s)
402	NdbfsDumpOpenFiles	NDBFS

Description. Prints list of NDB file system open files.

Sample Output.

```
Node 2: NDBFS: Dump open files: 10
Node 2:  0 (0x8792f70): /usr/local/mysql/cluster/ndb_2_fs/D1/DBDIH/P0.sysfile
Node 2:  1 (0x8794590): /usr/local/mysql/cluster/ndb_2_fs/D2/DBDIH/P0.sysfile
Node 2:  2 (0x878ed10): /usr/local/mysql/cluster/ndb_2_fs/D8/DBLQH/S0.FragLog
Node 2:  3 (0x8790330): /usr/local/mysql/cluster/ndb_2_fs/D9/DBLQH/S0.FragLog
Node 2:  4 (0x8791950): /usr/local/mysql/cluster/ndb_2_fs/D10/DBLQH/S0.FragLog
Node 2:  5 (0x8795da0): /usr/local/mysql/cluster/ndb_2_fs/D11/DBLQH/S0.FragLog
Node 2:  6 (0x8797358): /usr/local/mysql/cluster/ndb_2_fs/D8/DBLQH/S1.FragLog
Node 2:  7 (0x8798978): /usr/local/mysql/cluster/ndb_2_fs/D9/DBLQH/S1.FragLog
Node 2:  8 (0x8799f98): /usr/local/mysql/cluster/ndb_2_fs/D10/DBLQH/S1.FragLog
Node 2:  9 (0x879b5b8): /usr/local/mysql/cluster/ndb_2_fs/D11/DBLQH/S1.FragLog
```

Additional Information. [N/A]

2.18 DUMP 403

Code	Symbol	Kernel Block(s)
403	NdbfsDumpIdleFiles	NDBFS

Description. Prints list of NDB file system idle file handles.

Sample Output.

```
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: NDBFS: Dump idle files: 18
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  0 (0x7f5aec0029f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  1 (0x7f5aec0100f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  2 (0x7f5aec01d780): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  3 (0x7f5aec02add0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  4 (0x7f5aec0387f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  5 (0x7f5aec045e40): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  6 (0x7f5aec053490): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  7 (0x7f5aec060ae0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  8 (0x7f5aec06e130): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5:  9 (0x7f5aec07b780): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 10 (0x7f5aec088dd0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 11 (0x7f5aec0969f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 12 (0x7f5aec0a4040): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 13 (0x7f5aec0b1690): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 14 (0x7f5aec0bece0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 15 (0x7f5aec0cc330): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 16 (0x7f5aec0d9980): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 5: 17 (0x7f5aec0e6fd0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: NDBFS: Dump idle files: 18
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  0 (0x7fa0300029f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  1 (0x7fa0300100f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  2 (0x7fa03001d780): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  3 (0x7fa03002add0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  4 (0x7fa0300387f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  5 (0x7fa030045e40): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  6 (0x7fa030053490): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  7 (0x7fa030060ae0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  8 (0x7fa03006e130): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6:  9 (0x7fa03007b780): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 10 (0x7fa030088dd0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 11 (0x7fa0300969f0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 12 (0x7fa0300a4040): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 13 (0x7fa0300b1690): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 14 (0x7fa0300bece0): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 15 (0x7fa0300cc330): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 16 (0x7fa0300d9980): CLOSED
2014-10-13 21:18:48 [MgmtSrvr] INFO     -- Node 6: 17 (0x7fa0300e6fd0): CLOSED
```

Additional Information. [N/A]

2.19 DUMP 404 (OBSOLETE)

Code	Symbol	Kernel Block(s)
404	[none]	NDBFS

Description. Kills node or nodes. No longer used.

Sample Output.

```
...
```

Additional Information. [N/A]

2.20 DUMP 908

Code	Symbol	Kernel Block(s)
908	[none]	DBDIH, QMGR

Description. Causes heartbeat transmission information.to be written to the data node logs. Useful in conjunction with setting the `HeartbeatOrder` parameter.

Sample Output.

```
HB: pres:5 own:5 dyn:1-0 mxdyn:2 hb:6->5->6 node:dyn-hi,cfg: 5:1-0,0 6:2-0,0
```

Additional Information. [N/A]

2.21 DUMP 1000

Code	Symbol	Kernel Block(s)
1000	`DumpPageMemory`	`DBACC`, `DBTUP`

Description. Prints data node memory usage (`ACC` and `TUP`), as both a number of data pages, and the percentage of `DataMemory` and `IndexMemory` used.

Sample Output.

```
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Data usage is 0%(12 32K pages of total 32768)
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Index usage is 0%(24 8K pages of total 131104)
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 0 min: 34186 max: 74615 curr: 36334
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 3 min: 65544 max: 65544 curr: 32788
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 4 min: 720 max: 720 curr: 100
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 5 min: 1152 max: 1152 curr: 1088
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 6 min: 800 max: 1000 curr: 117
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 7 min: 2240 max: 2240 curr: 2240
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 5: Resource 9 min: 64 max: 0 curr: 1
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Data usage is 0%(12 32K pages of total 32768)
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Index usage is 0%(24 8K pages of total 131104)
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 0 min: 34207 max: 74615 curr: 36313
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 3 min: 65544 max: 65544 curr: 32788
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 4 min: 720 max: 720 curr: 95
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 5 min: 1152 max: 1152 curr: 1088
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 6 min: 800 max: 1000 curr: 102
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 7 min: 2240 max: 2240 curr: 2240
2014-10-15 12:06:29 [MgmtSrvr] INFO     -- Node 6: Resource 9 min: 64 max: 0 curr: 1
```

Note

When invoked as `ALL DUMP 1000`, this command reports memory usage for each data node separately, in turn.

Additional Information. You can also use the `ndb_mgm` client command `REPORT MEMORYUSAGE` to obtain this information (see Commands in the NDB Cluster Management Client). You can also query the `memoryusage` table (in the `ndbinfo` database) for this information.

2.22 DUMP 1223

Code	Symbol	Kernel Block(s)
1223	[none]	`DBDICT`

Description. Formerly, this killed the node. In NDB Cluster 7.4 and later, it has no effect.

Sample Output.

```
...
```

Additional Information. [N/A]

2.23 DUMP 1224

Code	Symbol	Kernel Block(s)
1224	[none]	DBDICT

Description. Formerly, this killed the node. In NDB Cluster 7.4 and later, it has no effect.

Sample Output.

```
...
```

Additional Information. [N/A]

2.24 DUMP 1225

Code	Symbol	Kernel Block(s)
1225	---	DBDICT

Description. Formerly, this killed the node. In NDB Cluster 7.4, it has no effect.

Sample Output.

```
...
```

Additional Information. [N/A]

2.25 DUMP 1226

Code	Symbol	Kernel Block(s)
1226	---	DBDICT

Description. Prints pool objects to the cluster log.

Sample Output.

```
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 5: c_obj_pool: 1332 1319
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 5: c_opRecordPool: 256 256
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 5: c_rope_pool: 146785 146615
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 6: c_obj_pool: 1332 1319
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 6: c_opRecordPool: 256 256
2014-10-15 12:13:22 [MgmtSrvr] INFO     -- Node 6: c_rope_pool: 146785 146615
```

Additional Information. [N/A]

2.26 DUMP 1228

Code	Symbol	Kernel Block(s)
1228	DictLockQueue	DBDICT

Description. Dumps the contents of the NDB internal dictionary lock queue to the cluster log.

Sample Output.

```
2014-10-15 12:14:08 [MgmtSrvr] INFO     -- Node 5: DICT : c_sub_startstop _outstanding 0 _lock 000000000000
2014-10-15 12:14:08 [MgmtSrvr] INFO     -- Node 6: DICT : c_sub_startstop _outstanding 0 _lock 000000000000
```

Additional Information. [N/A]

2.27 DUMP 1229

Code	Symbol	Kernel Block(s)
1229	DictDumpGetTabInfoQueue	DBDICT

Description. Shows state of GETTABINFOREQ queue.

Sample Output.

```
ndb_mgm> ALL DUMP 1229
Sending dump signal with data:
0x000004cd
Sending dump signal with data:
0x000004cd
```

Additional Information. Full debugging output requires the relevant data nodes to be configured with DictTrace >= 2 and relevant API nodes with ApiVerbose >= 2. See the descriptions of these parameters for more information.

Added in NDB 7.4.12 and NDB 7.5.2. (Bug #20368450)

2.28 DUMP 1332

Code	Symbol	Kernel Block(s)
1332	LqhDumpAllDefinedTabs	DBACC

Description. Prints the states of all tables known by the local query handler (LQH) to the cluster log.

Sample Output.

```
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 2 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 3 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 4 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 5 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 6 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
```

```
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 7 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 8 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 9 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 10 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5: Table 11 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 5:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 2 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 3 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 4 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 5 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 6 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 7 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 8 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 9 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 10 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6: Table 11 Status: 0 Usage: [ r: 0 w: 0 ]
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 0 distKey: 0
2014-10-15 12:15:07 [MgmtSrvr] INFO     -- Node 6:   frag: 1 distKey: 0
```

Additional Information. [N/A]

2.29 DUMP 1333

Code	Symbol	Kernel Block(s)
1333	LqhDumpNoLogPages	DBACC

Description. Reports redo log buffer usage.

Sample Output.

```
2014-10-15 12:16:05 [MgmtSrvr] INFO     -- Node 5: LQH: Log pages : 1024 Free: 960
2014-10-15 12:16:05 [MgmtSrvr] INFO     -- Node 6: LQH: Log pages : 1024 Free: 960
```

Additional Information. The redo log buffer is measured in 32KB pages, so the sample output can be interpreted as follows:

- **Redo log buffer total.** 1024 * 32K = 32MB
- **Redo log buffer free.** 960 * 32KB = ~31,457KB = ~30MB
- **Redo log buffer used.** (1024 - 960) * 32K = 2,097KB = ~2MB

2.30 DUMP 2300

Code	Symbol	Kernel Block(s)
2300	LqhDumpOneScanRec	DBLQH

Description. Prints the given scan record. Syntax: `DUMP 2300 recordno`.

Sample Output.

```
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5: Dblqh::ScanRecord[1]: state=0, type=0, complStatus=0, s
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  apiBref=0x2f40005, scanAccPtr=0
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  copyptr=-256, ailen=6, complOps=0, concurrOps=16
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  errCnt=0, schV=1
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  stpid=0, flag=2, lhold=0, lmode=0, num=134
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  relCount=16, TCwait=0, TCRec=3, KIflag=0
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 5:  LcpScan=1  RowId(0:0)
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[1]: state=0, type=0, complStatus=0, s
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x2f40006, scanAccPtr=0
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  copyptr=-256, ailen=6, complOps=0, concurrOps=16
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=1
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=2, lhold=0, lmode=0, num=134
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  relCount=16, TCwait=0, TCRec=2, KIflag=0
2014-10-15 12:33:35 [MgmtSrvr] INFO     -- Node 6:  LcpScan=1  RowId(0:0)
```

Additional Information. [N/A]

2.31 DUMP 2301

Code	Symbol	Kernel Block(s)
2301	LqhDumpAllScanRec	DBLQH

Description. Dump all scan records to the cluster log.

Sample Output. Only the first few scan records printed to the log for a single data node are shown here.

```
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: LQH: Dump all ScanRecords - size: 514
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[1]: state=0, type=0, complStatus=0, s
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x2f40006, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=-256, ailen=6, complOps=0, concurrOps=16
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=1
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=2, lhold=0, lmode=0, num=134
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=16, TCwait=0, TCRec=2, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=1  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[2]: state=0, type=0, complStatus=0, s
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
```

```
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[3]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[4]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[5]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[6]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[7]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6: Dblqh::ScanRecord[8]: state=0, type=0, complStatus=0, scanN
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  apiBref=0x0, scanAccPtr=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  copyptr=0, ailen=0, complOps=0, concurrOps=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  errCnt=0, schV=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  stpid=0, flag=0, lhold=0, lmode=0, num=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  relCount=0, TCwait=0, TCRec=0, KIflag=0
2014-10-15 12:40:00 [MgmtSrvr] INFO     -- Node 6:  LcpScan=0  RowId(0:0)
...
```

Additional Information. This `DUMP` code should be used sparingly if at all on an NDB Cluster in production, since hundreds or even thousands of scan records may be created on even a relatively small cluster that is not under load. For this reason, it is often preferable to print a single scan record using `DUMP 2300`.

The first line provides the total number of scan records dumped for this data node.

2.32 DUMP 2302

Code	Symbol	Kernel Block(s)
2302	`LqhDumpAllActiveScanRec`	`DBLQH`

Description. Dump only the active scan records from this node to the cluster log.

Sample Output.

```
2014-10-15 12:59:27 [MgmtSrvr] INFO     -- Node 5: LQH: Dump active ScanRecord - size: 514
```

```
2014-10-15 12:59:27 [MgmtSrvr] INFO     -- Node 6: LQH: Dump active ScanRecord - size: 514
...
```

Additional Information. The first line in each block of output contains the total number of (active and inactive) scan records. If nothing else is written to the log, then no scan records are currently active.

2.33 DUMP 2303

Code	Symbol	Kernel Block(s)
2303	`LqhDumpLcpState`	`DBLQH`

Description. Dumps the status of a local checkpoint from the point of view of a DBLQH block instance.

Beginning with NDB 7.2.6, this command also dumps the status of the single fragment scan record reserved for this LCP. (Bug #13986128)

Sample Output.

```
2014-10-15 13:01:37 [MgmtSrvr] INFO     -- Node 5: Local checkpoint 173 started. Keep GCI = 270929 oldest
2014-10-15 13:01:38 [MgmtSrvr] INFO     -- Node 5: LDM instance 1: Completed LCP, num fragments = 16 num re
2014-10-15 13:01:38 [MgmtSrvr] INFO     -- Node 6: LDM instance 1: Completed LCP, num fragments = 16 num re
2014-10-15 13:01:38 [MgmtSrvr] INFO     -- Node 5: Local checkpoint 173 completed
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5: == LQH LCP STATE ==
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5:  clcpCompletedState=0, c_lcpId=173, cnoOfFragsCheckpoint
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5:  lcpState=0 lastFragmentFlag=0
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5: currentFragment.fragPtrI=15
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5: currentFragment.lcpFragOrd.tableId=10
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5:  numFragLcpsQueued=0 reportEmpty=0
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 5:  m_EMPTY_LCP_REQ=0000000000000000
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6: == LQH LCP STATE ==
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6:  clcpCompletedState=0, c_lcpId=173, cnoOfFragsCheckpoint
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6:  lcpState=0 lastFragmentFlag=0
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6: currentFragment.fragPtrI=15
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6: currentFragment.lcpFragOrd.tableId=10
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6:  numFragLcpsQueued=0 reportEmpty=0
2014-10-15 13:02:04 [MgmtSrvr] INFO     -- Node 6:  m_EMPTY_LCP_REQ=0000000000000000
```

Additional Information. [N/A]

2.34 DUMP 2304

Code	Symbol	Kernel Block(s)
2304	[none]	`DBLQH`

Description. This command causes all fragment log files and their states to be written to the data node's out file (in the case of the data node having the node ID `5`, this would be `ndb_5_out.log`). The number of fragment log files is controlled by the `NoOfFragmentLogFiles` data node configuration parameter.

Sample Output. The following is taken from `ndb_5_out.log` in an NDB Cluster having 2 data nodes:

```
LP 0 blockInstance: 1 partNo: 0 state: 0 WW_Gci: 1 gcprec: -256 flq: 4294967040 4294967040 currfile: 0 tai
  file 0(0)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 2  currentFilepage 75
  file 1(1)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 0  currentFilepage 0
  file 2(2)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
```

```
  file 3(3)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 4(4)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 5(5)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 6(6)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 7(7)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 8(8)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 9(9)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 10(10)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 11(11)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 12(12)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 13(13)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 14(14)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 15(15)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
LP 1 blockInstance: 1 partNo: 1 state: 0 WW_Gci: 1 gcprec: -256 flq: 4294967040 4294967040 currfile: 16 tailFi
  file 0(16)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 2  currentFilepage 69
  file 1(17)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 0  currentFilepage 0
  file 2(18)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 3(19)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 4(20)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 5(21)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 6(22)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 7(23)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 8(24)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 9(25)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 10(26)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 11(27)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 12(28)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 13(29)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 14(30)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 15(31)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
LP 2 blockInstance: 1 partNo: 2 state: 0 WW_Gci: 1 gcprec: -256 flq: 4294967040 4294967040 currfile: 32 tailFi
  file 0(32)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 2  currentFilepage 69
  file 1(33)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 0  currentFilepage 0
  file 2(34)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 3(35)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 4(36)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 5(37)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 6(38)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 7(39)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 8(40)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 9(41)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 10(42)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 11(43)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 12(44)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 13(45)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 14(46)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 15(47)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
LP 3 blockInstance: 1 partNo: 3 state: 0 WW_Gci: 1 gcprec: -256 flq: 4294967040 4294967040 currfile: 48 tailFi
  file 0(48)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 2  currentFilepage 69
  file 1(49)  FileChangeState: 0  logFileStatus: 20  currentMbyte: 0  currentFilepage 0
  file 2(50)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 3(51)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 4(52)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 5(53)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 6(54)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 7(55)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 8(56)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 9(57)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 10(58)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 11(59)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 12(60)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 13(61)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 14(62)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
  file 15(63)  FileChangeState: 0  logFileStatus: 1  currentMbyte: 0  currentFilepage 0
```

Additional Information. The next 2 tables provide information about file change state codes and log file status codes as shown in the previous example.

FileChangeState Codes

Value	File Change State
0	Content row 1, column 2
1	NOT_ONGOING
2	BOTH_WRITES_ONGOING
3	LAST_WRITE_ONGOING
4	WRITE_PAGE_ZERO_ONGOING

LogFileStatus Codes

Value	Log File Status	Description
0	LFS_IDLE	Log file record not in use
1	CLOSED	Log file closed.
2	OPENING_INIT	---
3	OPEN_SR_FRONTPAGE	Log file opened as part of system restart; open file 0 to find the front page of the log part.
4	OPEN_SR_LAST_FILE	Opening last log file that was written before the system restart.
5	OPEN_SR_NEXT_FILE	Opening log file which is 16 files back (to find next available information about GCPs).
6	OPEN_EXEC_SR_START	Log file opened while executing the log during a system restart.
7	OPEN_EXEC_SR_NEW_MBYTE	---
8	OPEN_SR_FOURTH_PHASE	---
9	OPEN_SR_FOURTH_NEXT	---
10	OPEN_SR_FOURTH_ZERO	---
11	OPENING_WRITE_LOG	Log file opened while writing log (normal operation).
12	OPEN_EXEC_LOG	---
13	CLOSING_INIT	---
14	CLOSING_SR	Log file closed as part of system restart. Currently trying to find where to start executing the log.
15	CLOSING_EXEC_SR	Log file closed as part of log execution during system restart.
16	CLOSING_EXEC_SR_COMPLETED	---
17	CLOSING_WRITE_LOG	Log file closed as part of writing log during normal operations.
18	CLOSING_EXEC_LOG	---
19	OPEN_INIT	---
20	OPEN	Log file open.
21	OPEN_SR_READ_INVALIDATE_PAGES	---
22	CLOSE_SR_READ_INVALIDATE_PAGES	---

Value	Log File Status	Description
23	OPEN_SR_WRITE_INVALIDATE_PAGES	
24	CLOSE_SR_WRITE_INVALIDATE_PAGES	
25	OPEN_SR_READ_INVALIDATE_SEARCH_FILES	
26	CLOSE_SR_READ_INVALIDATE_SEARCH_FILES	
27	CLOSE_SR_READ_INVALIDATE_SEARCH_LAST_FILE	
28	OPEN_EXEC_LOG_CACHED	---
29	CLOSING_EXEC_LOG_CACHED	

More information about how these codes are defined can be found in the source file `storage/ndb/src/kernel/blocks/dblqh/Dblqh.hpp`. See also Section 2.35, "DUMP 2305".

2.35 DUMP 2305

Code	Symbol	Kernel Block(s)
2305	---	DBLQH

Description. Show the states of all fragment log files (see Section 2.34, "DUMP 2304"), then kills the node.

Sample Output.

```
. . .
```

Additional Information. [N/A]

2.36 DUMP 2308

Code	Symbol	Kernel Block(s)
2308	---	DBLQH

Description. Kills the node.

Sample Output.

```
. . .
```

Additional Information. [N/A]

2.37 DUMP 2315

Code	Symbol	Kernel Block(s)
2315	LqhErrorInsert5042	DBLQH

Description. [Unknown]

Sample Output. [N/A]

Additional Information. [N/A]

2.38 DUMP 2350

Code	Symbol	Kernel Block(s)
data_node_id 2350 *operation_filter*+	---	---

Description. Dumps all operations on a given data node or data nodes, according to the type and other parameters defined by the operation filter or filters specified.

Sample Output. Dump all operations on data node 2, from API node 5:

```
ndb_mgm> 2 DUMP 2350 1 5
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: Starting dump of operations
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: OP[470]:
Tab: 4 frag: 0 TC: 3 API: 5(0x8035)transid: 0x31c 0x3500500 op: SCAN state: InQueue
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: End of operation dump
```

Additional information. Information about operation filter and operation state values follows.

Operation filter values. The operation filter (or filters) can take on the following values:

Value	Filter
0	table ID
1	API node ID
2	2 transaction IDs, defining a range of transactions
3	transaction coordinator node ID

In each case, the ID of the object specified follows the specifier. See the sample output for examples.

Operation states. The “normal” states that may appear in the output from this command are listed here:

- *Transactions*:
  - `Prepared`: The transaction coordinator is idle, waiting for the API to proceed
  - `Running`: The transaction coordinator is currently preparing operations
  - `Committing`, `Prepare to commit`, `Commit sent`: The transaction coordinator is committing
  - `Completing`: The transaction coordinator is completing the commit (after commit, some cleanup is needed)
  - `Aborting`: The transaction coordinator is aborting the transaction
  - `Scanning`: The transaction coordinator is scanning
- *Scan operations*:
  - `WaitNextScan`: The scan is idle, waiting for API
  - `InQueue`: The scan has not yet started, but rather is waiting in queue for other scans to complete

- *Primary key operations*:
  - `In lock queue`: The operation is waiting on a lock
  - `Running`: The operation is being prepared
  - `Prepared`: The operation is prepared, holding an appropriate lock, and waiting for commit or rollback to complete

Relation to NDB API. It is possible to match the output of `DUMP 2350` to specific threads or `Ndb` objects. First suppose that you dump all operations on data node 2 from API node 5, using table 4 only, like this:

```
ndb_mgm> 2 DUMP 2350 1 5 0 4
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: Starting dump of operations
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: OP[470]:
Tab: 4 frag: 0 TC: 3 API: 5(0x8035)transid: 0x31c 0x3500500 op: SCAN state: InQueue
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: End of operation dump
```

Suppose you are working with an `Ndb` instance named `MyNdb`, to which this operation belongs. You can see that this is the case by calling the `Ndb` object's `getReference()` method, like this:

```
  printf("MyNdb.getReference(): 0x%x\n", MyNdb.getReference());
```

The output from the preceding line of code is:

```
MyNdb.getReference(): 0x80350005
```

The high 16 bits of the value shown corresponds to the number in parentheses from the `OP` line in the `DUMP` command's output (8035). For more about this method, see Ndb::getReference().

2.39 DUMP 2352

Code	Symbol	Kernel Block(s)
`node_id` 2352 *`operation_id`*	`---`	`---`

Description. Gets information about an operation with a given operation ID.

Sample Output. First, obtain a dump of operations. Here, we use `DUMP 2350` to get a dump of all operations on data node 5 from API node 100:

```
2014-10-15 13:36:26 [MgmtSrvr] INFO     -- Node 100: Event buffer status: used=1025KB(100%) alloc=1025KB(0%) m
```

In this case, there is a single operation reported on node 2, whose operation ID is 3. To obtain the transaction ID and primary key for the operation having 3 as its ID, we use the node ID and operation ID with `DUMP 2352` as shown here:

```
ndb_mgm> 5 DUMP 2352 3
```

The following is written to the cluster log:

```
```

```
2014-10-15 13:45:20 [MgmtSrvr] INFO     -- Node 5: OP[3]: transid: 0xf 0x806400 key: 0x2
```

Additional Information. Use `DUMP 2350` to obtain an operation ID. See Section 2.38, "DUMP 2350", and the previous example.

2.40 DUMP 2354

Code	Symbol	Kernel Block(s)
2354	`LqhReportCopyInfo`	`DBLQH`

Description. Prints a given scan fragment record, given the instance. The syntax is shown here:

```
DUMP 2354 recordno instanceno
```

Here, *`recordno`* is the scan fragment record number, and `instanceno` is the number of the instance.

Sample Output.

```
2014-10-13 16:30:57 [MgmtSrvr] INFO     -- Node 5: LDM instance 1: CopyFrag complete. 0 frags, +0/-0 rows,
2014-10-13 16:30:57 [MgmtSrvr] INFO     -- Node 6: LDM instance 1: CopyFrag complete. 0 frags, +0/-0 rows,
```

Additional Information. This `DUMP` code was added in NDB 7.4.1.

2.41 DUMP 2398

Code	Symbol	Kernel Block(s)
`node_id` 2398	[none]	`DBLQH`

Description. Dumps information about free space in log part files for the data node with the node ID *`node_id`*. The dump is written to the data node out log rather than to the cluster log.

Sample Output. Written to `ndb_6_out.log`:

```
REDO part: 0 HEAD: file: 0 mbyte: 2 TAIL: file: 0 mbyte: 2 total: 256 free: 256 (mb)
REDO part: 1 HEAD: file: 0 mbyte: 2 TAIL: file: 0 mbyte: 2 total: 256 free: 256 (mb)
REDO part: 2 HEAD: file: 0 mbyte: 2 TAIL: file: 0 mbyte: 2 total: 256 free: 256 (mb)
REDO part: 3 HEAD: file: 0 mbyte: 2 TAIL: file: 0 mbyte: 2 total: 256 free: 256 (mb)
```

Additional Information. Each line of the output has the following format (shown here split across two lines for legibility):

```
REDO part: part_no HEAD: file: start_file_no mbyte: start_pos
TAIL: file: end_file_no mbyte: end_pos total: total_space free: free_space (mb)
```

A data node's redo log is divided into four parts; thus, *`part_no`* is always a number between 0 and 3 inclusive. The parts are stored in the data node file system `D8`, `D9`, `D10`, and `D11` directories with redo log part 0 being stored in `D8`, part 1 in `D9`, and so on (see Section 1.1.2, "NDB Cluster Data Node File System Directory Files"). Within each directory can be found a `DBLQH` subdirectory containing `NoOfFragmentLogFiles` files. The default value for `NoOfFragmentLogFiles` is 16. The default size of each of these files is 16 MB; this can be changed by setting the `FragmentLogFileSize` configuration parameter.

start_file_no indicates the number of the file and *start_pos* the point inside this file in which the redo log starts; for the example just shown, since *part_no* is 0, this means that the redo log starts at approximately 12 MB from the end of the file D8/DBLQH/S6.FragLog.

Similarly, *end_file_no* corresponds to the number of the file and *end_pos* to the point within that file where the redo log ends. Thus, in the previous example, the redo log's end point comes approximately 10 MB from the end of D8/DBLQH/S6.FragLog.

total_space shows the total amount of space reserved for part *part_no* of the redo log. This is equal to NoOfFragmentLogFiles * FragmentLogFileSize; by default this is 16 times 16 MB, or 256 MB. *free_space* shows the amount remaining. Thus, the amount used is equal to *total_space* - *free_space*; in this example, this is 256 - 254 = 2 MB.

Caution

It is not recommended to execute DUMP 2398 while a data node restart is in progress.

2.42 DUMP 2399

Code	Symbol	Kernel Block(s)
node_id 2399	[none]	DBLQH

Description. Similarly to DUMP 2398, this command dumps information about free space in log part files for the data node with the node ID *node_id*. Unlike the case with DUMP 2398, the dump is written to the cluster log, and includes a figure for the percentage of free space remaining in the redo log.

Sample Output.

```
ndb_mgm> 6 DUMP 2399
Sending dump signal with data:
0x0000095f
```

(Written to cluster log:)

```
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 5: Logpart: 0 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 5: Logpart: 1 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 5: Logpart: 2 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 5: Logpart: 3 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 6: Logpart: 0 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 6: Logpart: 1 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 6: Logpart: 2 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
2014-10-15 13:39:50 [MgmtSrvr] INFO     -- Node 6: Logpart: 3 head=[ file: 0 mbyte: 2 ] tail=[ file: 0 mbyte:
```

Additional Information. Each line of the output uses the following format (shown here split across two lines for legibility):

```
timestamp [MgmtSrvr] INFO     -- Node node_id: Logpart: part_no head=[ file: start_file_no mbyte: start_pos ]
tail=[ file: end_file_no mbyte: end_pos ] total mb: total_space free mb: free_space free%: free_pct
```

timestamp shows when the command was executed by data node *node_id*. A data node's redo log is divided into four parts. which part is indicated by *part_no* (always a number between 0 and 3 inclusive). The parts are stored in the data node file system directories named D8, D9, D10, and D11; redo log part 0 is stored in D8, part 1 in D9, and so on. Within each of these four directories is a DBLQH subdirectory containing NoOfFragmentLogFiles fragment log files. The default value for

`NoOfFragmentLogFiles` is 16. The default size of each of these files is 16 MB; this can be changed by setting the `FragmentLogFileSize` configuration parameter. (See Section 1.1.2, “NDB Cluster Data Node File System Directory Files”, for more information about the fragment log files.)

`start_file_no` indicates the number of the file and *`start_pos`* the point inside this file in which the redo log starts; for the example just shown, since *`part_no`* is 0, this means that the redo log starts at approximately 12 MB from the end of the file `D8/DBLQH/S6.FragLog`.

Similarly, *`end_file_no`* corresponds to the number of the file and *`end_pos`* to the point within that file where the redo log ends. Thus, in the previous example, the redo log's end point comes approximately 10 MB from the end of `D8/DBLQH/S6.FragLog`.

`total_space` shows the total amount of space reserved for part *`part_no`* of the redo log. This is equal to `NoOfFragmentLogFiles * FragmentLogFileSize`; by default this is 16 times 16 MB, or 256 MB. *`free_space`* shows the amount remaining. The amount used is equal to *`total_space - free_space`*; in this example, this is 256 - 254 = 2 MB. *`free_pct`* shows the ratio of *`free_space`* to *`total_space`*, expressed as whole-number percentage. In the example just shown, this is equal to 100 * (254 / 256), or approximately 99 percent.

2.43 DUMP 2400

Code	Symbol	Kernel Block(s)
2400 *`record_id`*	`AccDumpOneScanRec`	`DBACC`

Description. Dumps the scan record having record ID *`record_id`*.

Sample Output. From `ALL DUMP 2400 1` the following output is written to the cluster log:

```
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 5: Dbacc::ScanRec[1]: state=1, transid(0x0, 0x0)
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 5:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 5:  scanNextfreerec=2 firstActOp=0 firstLockedOp=0, scanLas
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 5:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, r
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 5:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scan
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 6: Dbacc::ScanRec[1]: state=1, transid(0x0, 0x0)
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 6:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 6:  scanNextfreerec=2 firstActOp=0 firstLockedOp=0, scanLas
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 6:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, r
2014-10-15 13:49:50 [MgmtSrvr] INFO     -- Node 6:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scan
```

Additional Information. For dumping all scan records, see Section 2.44, “DUMP 2401”.

2.44 DUMP 2401

Code	Symbol	Kernel Block(s)
2401	`AccDumpAllScanRec`	`DBACC`

Description. Dumps all scan records for the node specified.

Sample Output.

```
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5: ACC: Dump all ScanRec - size: 514
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5: Dbacc::ScanRec[1]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanNextfreerec=2 firstActOp=0 firstLockedOp=0, scanLas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, r
```

```
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5: Dbacc::ScanRec[2]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanNextfreerec=3 firstActOp=0 firstLockedOp=0, scanLastLo
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, maxB
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5: Dbacc::ScanRec[3]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanNextfreerec=4 firstActOp=0 firstLockedOp=0, scanLastLo
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, maxB
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 5:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas

.
.
.

2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6: Dbacc::ScanRec[511]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanNextfreerec=512 firstActOp=0 firstLockedOp=0, scanLast
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, maxB
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6: Dbacc::ScanRec[512]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanNextfreerec=513 firstActOp=0 firstLockedOp=0, scanLast
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, maxB
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6: Dbacc::ScanRec[513]: state=1, transid(0x0, 0x0)
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  activeLocalFrag=0, nextBucketIndex=0
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanNextfreerec=-256 firstActOp=0 firstLockedOp=0, scanLas
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanUserP=0, startNoBuck=0, minBucketIndexToRescan=0, maxB
2014-10-15 13:52:06 [MgmtSrvr] INFO     -- Node 6:  scanBucketState=0, scanLockHeld=0, userBlockRef=0, scanMas
```

Additional Information. Use this command with caution, as there may be a great many scans. If you want to dump a single scan record, given its record ID, see Section 2.43, “DUMP 2400”; for dumping all active scan records, see Section 2.45, “DUMP 2402”.

2.45 DUMP 2402

Code	Symbol	Kernel Block(s)
2402	`AccDumpAllActiveScanRec`	`DBACC`

Description. Dumps all active scan records.

Sample Output. Similar to that for DUMP 2400 and DUMP 2401. See Section 2.44, “DUMP 2401”.

Additional Information. To dump all scan records (active or not), see Section 2.44, “DUMP 2401”.

2.46 DUMP 2403

Code	Symbol	Kernel Block(s)
2403 *`record_id`*	`AccDumpOneOperationRec`	`DBACC`

Description. Dumps a given operation record, given its ID. No arguments other than this (and the node ID or `ALL`) are required.

Sample Output. (For `ALL DUMP 2403 1`:)

```
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: Dbacc::operationrec[1]: transid(0x0, 0x306400)
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: elementIsforward=1, elementPage=131095, elementPointer=118
```

```
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: fid=0, fragptr=8
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: hashValue=-946144765
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: nextLockOwnerOp=-256, nextOp=-256, nextParallelQue=2
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: nextSerialQue=-256, prevOp=0
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: prevLockOwnerOp=-256, prevParallelQue=2
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: prevSerialQue=-256, scanRecPtr=-256
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 5: m_op_bits=0xffffffff, scanBits=0, reducedHashValue=ebe8
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: Dbacc::operationrec[1]: transid(0xf, 0x806400)
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: elementIsforward=1, elementPage=131078, elementPointer=
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: fid=1, fragptr=17
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: hashValue=-498516881
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: nextLockOwnerOp=-256, nextOp=-256, nextParallelQue=-256
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: nextSerialQue=-256, prevOp=0
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: prevLockOwnerOp=4, prevParallelQue=-256
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: prevSerialQue=-256, scanRecPtr=-256
2014-10-15 13:56:26 [MgmtSrvr] INFO     -- Node 6: m_op_bits=0xffffffff, scanBits=0, reducedHashValue=a4f1
```

Additional Information. [N/A]

2.47 DUMP 2404

Code	Symbol	Kernel Block(s)
2404	`AccDumpNumOpRecs`	`DBACC`

Description. Prints the number of operation records (total number, and number free) to the cluster log.

Sample Output.

```
2014-10-15 13:59:27 [MgmtSrvr] INFO     -- Node 5: Dbacc::OperationRecords: num=167764, free=131670
2014-10-15 13:59:27 [MgmtSrvr] INFO     -- Node 6: Dbacc::OperationRecords: num=167764, free=131670
```

Additional Information. The total number of operation records is determined by the value set for the `MaxNoOfConcurrentOperations` configuration parameter.

2.48 DUMP 2405

Code	Symbol	Kernel Block(s)
2405	`AccDumpFreeOpRecs`	

Description. Unknown: No output results if this command is called without additional arguments; if an extra argument is used, this command crashes the data node.

Sample Output. (For `2 DUMP 2405 1`:)

```
Time: Sunday 01 November 2015 - 18:33:54
Status: Temporary error, restart node
Message: Job buffer congestion (Internal error, programming error or
missing error message, please report a bug)
Error: 2334
Error data: Job Buffer Full
Error object: APZJobBuffer.C
Program: ./libexec/ndbd
Pid: 27670
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.49 DUMP 2406

Code	Symbol	Kernel Block(s)
2406	AccDumpNotFreeOpRecs	DBACC

Description. Unknown: No output results if this command is called without additional arguments; if an extra argument is used, this command crashes the data node.

Sample Output. (For `2 DUMP 2406 1`:)

```
Time: Sunday 01 November 2015 - 18:39:16
Status: Temporary error, restart node
Message: Job buffer congestion (Internal error, programming error or
missing error message, please report a bug)
Error: 2334
Error data: Job Buffer Full
Error object: APZJobBuffer.C
Program: ./libexec/ndbd
Pid: 27956
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.50 DUMP 2500

In NDB Cluster 7.4 and later, this DUMP code prints a set of scan fragment records to the cluster log.

Code	Symbol	Kernel Block(s)
2500	TcDumpSetOfScanFragRec	DBTC

Description. This DUMP code uses the syntax shown here:

```
DUMP 2500 recordno numrecords dbtcinst [activeonly]
```

This prints *`numrecords`* records from DBTC instance *`dbtcinst`*, starting with the record having record number *`recno`*. The last argument is optional; all of the others shown are required. *`activeonly`* is a boolean that determines whether or not to print only active records. If set to 1 (actually, any nonzero value), only active records are printed and ignore any free records not in use for the moment. 0 means all records are included. The default is 1.

Sample Output.

```
. . .
```

Additional Information. [N/A]

Prior to NDB Cluster 7.4, this DUMP code had a different symbol and function, as described in this table and the notes that follow.

Code	Symbol	Kernel Block(s)
2500	TcDumpAllScanFragRec	DBTC

Description. Kills the data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 13:37:11
Status: Temporary error, restart node
Message: Assertion (Internal error, programming error or missing error
message, please report a bug)
Error: 2301
Error data: ArrayPool<T>::getPtr
Error object: ../../../../../storage/ndb/src/kernel/vm/ArrayPool.hpp line: 345
(block: CMVMI)
Program: ./libexec/ndbd
Pid: 13237
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.21-ndb-7.3.7
```

2.51 DUMP 2501

Code	Symbol	Kernel Block(s)
2501	TcDumpOneScanFragRec	DBTC

Description. No output if called without any additional arguments. With additional arguments, it kills the data node.

Sample Output. (For `2 DUMP 2501 1`:)

```
Time: Sunday 01 November 2015 - 18:41:41
Status: Temporary error, restart node
Message: Assertion (Internal error, programming error or missing error
message, please report a bug)
Error: 2301
Error data: ArrayPool<T>::getPtr
Error object: ../../../../../storage/ndb/src/kernel/vm/ArrayPool.hpp line: 345
(block: DBTC)
Program: ./libexec/ndbd
Pid: 28239
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.52 DUMP 2502

In NDB Cluster 7.4 and later, this code can be used to print a set of scan records for a given DBTC block instance in the cluster log.

Code	Symbol	Kernel Block(s)
2502	TcDumpAllScanRec	DBTC

Description. This DUMP code uses the syntax shown here:

```
DUMP 2502 recordno numrecords dbtcinst [activeonly]
```

This prints *numrecords* scan records from DBTC instance number *dbtcinst*, starting with the record having record number *recno*. The last argument is optional; all of the others shown are required.

activeonly is a boolean that determines whether or not to print only active records. If set to 1 (actually, any nonzero value), only active records are printed and ignore any free records not in use for the moment. 0 means all records are included. The default is 1.

NDB Cluster 7.3 and earlier:

Code	Symbol	Kernel Block(s)
2502	TcDumpAllScanRec	DBTC

Description. Dumps all scan records held by TC blocks.

Sample Output.

```
Node 2: TC: Dump all ScanRecord - size: 256
Node 2: Dbtc::ScanRecord[1]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=2
Node 2: Dbtc::ScanRecord[2]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=3
Node 2: Dbtc::ScanRecord[3]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=4

.
.
.

Node 2: Dbtc::ScanRecord[254]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=255
Node 2: Dbtc::ScanRecord[255]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=-256
Node 2: Dbtc::ScanRecord[255]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=-256
```

Additional Information. [N/A]

2.53 DUMP 2503 (OBSOLETE)

Note

This DUMP code was removed in NDB 7.4.1.

Code	Symbol	Kernel Block(s)
2503	TcDumpAllActiveScanRec	DBTC

Description. Dumps all active scan records.

Sample Output.

```
Node 2: TC: Dump active ScanRecord - size: 256
```

Additional Information. [N/A]

2.54 DUMP 2504

Code	Symbol	Kernel Block(s)
2504 *record_id*	TcDumpOneScanRec	DBTC

Description. Dumps a single scan record having the record ID *record_id*. (For dumping all scan records, see Section 2.52, “DUMP 2502”.)

Sample Output. (For `2 DUMP 2504 1`:)

```
Node 2: Dbtc::ScanRecord[1]: state=0nextfrag=0, nofrag=0
Node 2:  ailen=0, para=0, receivedop=0, noOprePperFrag=0
Node 2:  schv=0, tab=0, sproc=0
Node 2:  apiRec=-256, next=2
```

Additional Information. The attributes in the output of this command are described as follows:

- **ScanRecord.** The scan record slot number (same as *record_id*)
- **state.** One of the following values (found as `ScanState` in `Dbtc.hpp`):

Value	State
0	IDLE
1	WAIT_SCAN_TAB_INFO
2	WAIT_AI
3	WAIT_FRAGMENT_COUNT
4	RUNNING
5	CLOSING_SCAN

- `nextfrag`: ID of the next fragment to be scanned. Used by a scan fragment process when it is ready for the next fragment.
- `nofrag`: Total number of fragments in the table being scanned.
- `ailen`: Length of the expected attribute information.
- `para`: Number of scan frag processes that belonging to this scan.
- `receivedop`: Number of operations received.
- `noOprePperFrag`: Maximum number of bytes per batch.
- `schv`: Schema version used by this scan.
- `tab`: The index or table that is scanned.
- `sproc`: Index of stored procedure belonging to this scan.
- `apiRec`: Reference to `ApiConnectRecord`

- `next`: Index of next `ScanRecord` in free list

2.55 DUMP 2505

Code	Symbol	Kernel Block(s)
2505	`TcDumpOneApiConnectRec`	`DBTC`

Description. Prints the API connection record *`recordno`* from instance *`instanceno`*, using the syntax shown here:

```
DUMP 2505 recordno instanceno
```

Sample Output.

```
...
```

Additional Information. `DUMP` code 2505 was added in NDB 7.4.1.

2.56 DUMP 2506 (OBSOLETE)

Note

This `DUMP` code was removed in NDB 7.4.1.

Code	Symbol	Kernel Block(s)
2506	`TcDumpAllApiConnectRec`	`DBTC`

Description. [Unknown]

Sample Output.

```
Node 2: TC: Dump all ApiConnectRecord - size: 12288
Node 2: Dbtc::ApiConnectRecord[1]: state=0, abortState=0, apiFailState=0
Node 2:  transid(0x0, 0x0), apiBref=0x1000002, scanRec=-256
Node 2:  ctcTimer=36057, apiTimer=0, counter=0, retcode=0, retsig=0
Node 2:  lqhkeyconfrec=0, lqhkeyreqrec=0, tckeyrec=0
Node 2:  next=-256
Node 2: Dbtc::ApiConnectRecord[2]: state=0, abortState=0, apiFailState=0
Node 2:  transid(0x0, 0x0), apiBref=0x1000002, scanRec=-256
Node 2:  ctcTimer=36057, apiTimer=0, counter=0, retcode=0, retsig=0
Node 2:  lqhkeyconfrec=0, lqhkeyreqrec=0, tckeyrec=0
Node 2:  next=-256
Node 2: Dbtc::ApiConnectRecord[3]: state=0, abortState=0, apiFailState=0
Node 2:  transid(0x0, 0x0), apiBref=0x1000002, scanRec=-256
Node 2:  ctcTimer=36057, apiTimer=0, counter=0, retcode=0, retsig=0
Node 2:  lqhkeyconfrec=0, lqhkeyreqrec=0, tckeyrec=0
Node 2:  next=-256

.
.
.

Node 2: Dbtc::ApiConnectRecord[12287]: state=7, abortState=0, apiFailState=0
Node 2:  transid(0x0, 0x0), apiBref=0xffffffff, scanRec=-256
Node 2:  ctcTimer=36308, apiTimer=0, counter=0, retcode=0, retsig=0
Node 2:  lqhkeyconfrec=0, lqhkeyreqrec=0, tckeyrec=0
Node 2:  next=-256
```

```
Node 2: Dbtc::ApiConnectRecord[12287]: state=7, abortState=0, apiFailState=0
Node 2:  transid(0x0, 0x0), apiBref=0xffffffff, scanRec=-256
Node 2:  ctcTimer=36308, apiTimer=0, counter=0, retcode=0, retsig=0
Node 2:  lqhkeyconfrec=0, lqhkeyreqrec=0, tckeyrec=0
Node 2:  next=-256
```

Additional Information. If the default settings are used, the output from this command is likely to exceed the maximum log file size.

2.57 DUMP 2507

Code	Symbol	Kernel Block(s)
2507	TcSetTransactionTimeout	DBTC

Description. Apparently requires an extra argument, but is not currently known with certainty.

Sample Output.

```
...
```

Additional Information. [N/A]

2.58 DUMP 2508

Code	Symbol	Kernel Block(s)
2508	TcSetApplTransactionTimeout	DBTC

Description. Apparently requires an extra argument, but is not currently known with certainty.

Sample Output.

```
...
```

Additional Information. [N/A]

2.59 DUMP 2509

Code	Symbol	Kernel Block(s)
2509	StartTcTimer	DBTC

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.60 DUMP 2510

Code	Symbol	Kernel Block(s)
2510	StopTcTimer	DBTC

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.61 DUMP 2511

Code	Symbol	Kernel Block(s)
2511	StartPeriodicTcTimer	DBTC

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.62 DUMP 2512

Code	Symbol	Kernel Block(s)
2512 [*delay*]	TcStartDumpIndexOpCount	DBTC

Description. Dumps the value of MaxNoOfConcurrentIndexOperations, and the current resource usage, in a continuous loop. The *delay* time between reports can optionally be specified (in seconds), with the default being 1 and the maximum value being 25 (values greater than 25 are silently coerced to 25).

Sample Output. (Single report:)

```
Node 2: IndexOpCount: pool: 8192 free: 8192
```

Additional Information. There appears to be no way to disable the repeated checking of MaxNoOfConcurrentIndexOperations once started by this command, except by restarting the data node. It may be preferable for this reason to use DUMP 2513 instead (see Section 2.63, “DUMP 2513”).

2.63 DUMP 2513

Code	Symbol	Kernel Block(s)
2513	TcDumpIndexOpCount	

Description. Dumps the value of MaxNoOfConcurrentIndexOperations, and the current resource usage.

Sample Output.

```
Node 2: IndexOpCount: pool: 8192 free: 8192
```

Additional Information. Unlike the continuous checking done by `DUMP 2512` the check is performed only once (see Section 2.62, “DUMP 2512”).

2.64 DUMP 2514

Code	Symbol	Kernel Block(s)
2514	`TcDumpApiConnectRecSummary`	`DBTC`

Description. Provides information counts for allocated, seized, stateless, stateful, and scanning transaction objects for each API node. Available beginning with NDB 7.2.13. (Bug #15878085)

The syntax for this command is shown here:

```
DUMP 2514 [instanceno]
```

This command takes the `DBTC` instance number (*`instanceno`*) as an optional argument; if not specified, it defaults to 0. The *`instanceno`* is not needed if there is only one instance of `DBTC`.

Sample Output.

```
Start of ApiConnectRec summary (6144 total allocated)
  Api node 10 connect records seized : 0 stateless : 0 stateful : 0 scan : 0
  Api node 11 connect records seized : 2 stateless : 0 stateful : 0 scan : 0
  Api node 12 connect records seized : 1 stateless : 0 stateful : 0 scan : 0
```

The total number of records allocated depends on the number of transactions and a number of other factors, with the value of `MaxNoOfConcurrentTransactions` setting an upper limit. See the description of this parameter for more information

Additional Information. There are two possible states for each record, listed here:

1. *Available*: In the per-data node pool, not yet seized by any API node
2. *Seized*: Reserved from the per-data node pool by a particular API

Seized nodes further be divided into a number of categories or sub-states, as shown in the following list:

- *Ready*: (Not counted here) Seized, ready for use; can be calculated for an API as # seized - (# stateless + # stateful + # scan)
- *Stateless*: Record was last used for a 'stateless' transaction, and is effectively ready
- *Stateful*: Record is in use by a transaction
- *Scan*: Record is in use for a scan (table or ordered index)

2.65 DUMP 2515

Code	Symbol	Kernel Block(s)
2515	`TcDumpSetOfApiConnectRec`	`DBTC`

Description. Prints a range of API connection records. The syntax is as shown here, where *`recordno`* is the number of the first record, *`numrecords`* is the number of records to be dumped, and *`instanceno`* is the block instance number:

```
DUMP 2515 recordno numrecords instanceno
```

Caution

It is recommended not to print more than 10 records at a time using this DUMP code from a cluster under load.

Sample Output. ...

```
...
```

Additional Information. DUMP code 2515 was added in NDB 7.4.1.

2.66 DUMP 2516

Code	Symbol	Kernel Block(s)
2516	TcDumpOneTcConnectRec	DBTC

Description. Prints the TC connection record *recordno* from instance *instanceno*, using the syntax shown here:

```
DUMP 2516 recordno instanceno
```

To print a series of such records, use DUMP 2517.

Sample Output.

```
...
```

Additional Information. DUMP code 2516 was added in NDB 7.4.1.

2.67 DUMP 2517

Code	Symbol	Kernel Block(s)
2515	TcDumpSetOfTcConnectRec	DBTC

Description. Prints a range of TC connection records. The syntax is as shown here, where *recordno* is the number of the first record, *numrecords* is the number of records to be dumped, and *instanceno* is the block instance number:

```
DUMP 2517 recordno numrecords instanceno
```

Caution

It is recommended not to print more than 10 records at a time using DUMP 2517 code from a cluster under load.

Sample Output. ...

```
...
```

Additional Information. DUMP code 2517 was added in NDB 7.4.1.

2.68 DUMP 2550

Code	Symbol	Kernel Block(s)
data_node_id 2550 *transaction_filter*+	---	---

Description. Dumps all transaction from data node *data_node_id* meeting the conditions established by the transaction filter or filters specified.

Sample Output. Dump all transactions on node 2 which have been inactive for 30 seconds or longer:

```
ndb_mgm> 2 DUMP 2550 4 30
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: Starting dump of transactions
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: TRX[123]: API: 5(0x8035) transid: 0x31c 0x3500500 inacti
2011-11-01 13:16:49 [MgmSrvr] INFO     -- Node 2: End of transaction dump
```

Additional Information. The following values may be used for transaction filters. The filter value must be followed by one or more node IDs or, in the case of the last entry in the table, by the time in seconds that transactions have been inactive:

Value	Filter
1	API node ID
2	2 transaction IDs, defining a range of transactions
4	time transactions inactive (seconds)

2.69 DUMP 2555

Code	Symbol	Kernel Block(s)
2555	TcDumpPoolLevels	DBTC

Description. Prints pool levels to the cluster log.

Sample Output.

```
...
```

Additional Information. This DUMP code was added in NDB 7.4.1.

2.70 DUMP 2600

Code	Symbol	Kernel Block(s)
2600	CmvmiDumpConnections	CMVMI

Description. Shows status of connections between all cluster nodes. When the cluster is operating normally, every connection has the same status.

Sample Output.

```
Node 3: Connection to 1 (MGM) is connected
```

```
Node 3: Connection to 2 (MGM) is trying to connect
Node 3: Connection to 3 (DB) does nothing
Node 3: Connection to 4 (DB) is connected
Node 3: Connection to 7 (API) is connected
Node 3: Connection to 8 (API) is connected
Node 3: Connection to 9 (API) is trying to connect
Node 3: Connection to 10 (API) is trying to connect
Node 3: Connection to 11 (API) is trying to connect
Node 4: Connection to 1 (MGM) is connected
Node 4: Connection to 2 (MGM) is trying to connect
Node 4: Connection to 3 (DB) is connected
Node 4: Connection to 4 (DB) does nothing
Node 4: Connection to 7 (API) is connected
Node 4: Connection to 8 (API) is connected
Node 4: Connection to 9 (API) is trying to connect
Node 4: Connection to 10 (API) is trying to connect
Node 4: Connection to 11 (API) is trying to connect
```

Additional Information. The message `is trying to connect` actually means that the node in question was not started. This can also be seen when there are unused `[api]` or `[mysql]` sections in the `config.ini` file nodes configured—in other words when there are spare slots for API or SQL nodes.

2.71 DUMP 2601

Code	Symbol	Kernel Block(s)
2601	`CmvmiDumpLongSignalMemory`	`CMVMI`

Description. [Unknown]

Sample Output.

```
Node 2: Cmvmi: g_sectionSegmentPool size: 4096 free: 4096
```

Additional Information. [N/A]

2.72 DUMP 2602

Code	Symbol	Kernel Block(s)
2602	`CmvmiSetRestartOnErrorInsert`	`CMVMI`

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.73 DUMP 2603

Code	Symbol	Kernel Block(s)
2603	`CmvmiTestLongSigWithDelay`	`CMVMI`

Description. [Unknown]

Sample Output.

```
. . .
```

Additional Information. [N/A]

2.74 DUMP 2604

Code	Symbol	Kernel Block(s)
2604	`CmvmiDumpSubscriptions`	`CMVMI`

Description. Dumps current event subscriptions.

Note

This output appears in the `ndb_node_id_out.log` file (local to each data node) and not in the management server (global) cluster log file.

Sample Output.

```
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- List subscriptions:
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Subscription: 0, nodeId: 1, ref: 0x80000001
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 0 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 1 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 2 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 3 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 4 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 5 Level 8
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 6 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 7 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 8 Level 15
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 9 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 10 Level 7
Sunday 01 November 2015 17:10:54 [ndbd] INFO     -- Category 11 Level 15
```

Additional Information. The output lists all event subscriptions; for each subscription a header line and a list of categories with their current log levels is printed. The following information is included in the output:

- `Subscription`: The event subscription's internal ID
- `nodeID`: Node ID of the subscribing node
- `ref`: A block reference, consisting of a block ID from `storage/ndb/include/kernel/BlockNumbers.h` shifted to the left by 4 hexadecimal digits (16 bits) followed by a 4-digit hexadecimal node number. Block id `0x8000` appears to be a placeholder; it is defined as `MIN_API_BLOCK_NO`, with the node number part being 1 as expected
- `Category`: The cluster log category, as listed in Event Reports Generated in NDB Cluster (see also the file `storage/ndb/include/mgmapi/mgmapi_config_parameters.h`).
- `Level`: The event level setting (the range being 0 to 15).

2.75 DUMP 5900

Code	Symbol	Kernel Block(s)
5900	`LCPContinue`	`DBLQH`

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.76 DUMP 7000

Code	Symbol	Kernel Block(s)
7000	---	DBDIH

Description. Prints information on GCP state

Sample Output.

```
Node 2: ctimer = 299072, cgcpParticipantState = 0, cgcpStatus = 0
Node 2: coldGcpStatus = 0, coldGcpId = 436, cmasterState = 1
Node 2: cmasterTakeOverNode = 65535, ctcCounter = 299072
```

Additional Information. [N/A]

2.77 DUMP 7001

Code	Symbol	Kernel Block(s)
7001	---	DBDIH

Description. Prints information on the current LCP state.

Sample Output.

```
Node 2: c_lcpState.keepGci = 1
Node 2: c_lcpState.lcpStatus = 0, clcpStopGcp = 1
Node 2: cgcpStartCounter = 7, cimmediateLcpStart = 0
```

Additional Information. [N/A]

2.78 DUMP 7002

Code	Symbol	Kernel Block(s)
7002	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: cnoOfActiveTables = 4, cgcpDelay = 2000
Node 2: cdictblockref = 16384002, cfailurenr = 1
Node 2: con_lineNodes = 2, reference() = 16121858, creceivedfrag = 0
```

Additional Information. [N/A]

2.79 DUMP 7003

Code	Symbol	Kernel Block(s)
7003	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: cfirstAliveNode = 2, cgckptflag = 0
Node 2: clocallqhblockref = 16187394, clocaltcblockref = 16056322, cgcpOrderBlocked = 0
Node 2: cstarttype = 0, csystemnodes = 2, currentgcp = 438
```

Additional Information. [N/A]

2.80 DUMP 7004

Code	Symbol	Kernel Block(s)
7004	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: cmasterdihref = 16121858, cownNodeId = 2, cnewgcp = 438
Node 2: cndbStartReqBlockref = 16449538, cremainingfrags = 1268
Node 2: cntrlblockref = 16449538, cgcpSameCounter = 16, coldgcp = 437
```

Additional Information. [N/A]

2.81 DUMP 7005

Code	Symbol	Kernel Block(s)
7005	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: crestartGci = 1
```

Additional Information. [N/A]

2.82 DUMP 7006

Code	Symbol	Kernel Block(s)
7006	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: clcpDelay = 20, cgcpMasterTakeOverState = 0
Node 2: cmasterNodeId = 2
Node 2: cnoHotSpare = 0, c_nodeStartMaster.startNode = -256, c_nodeStartMaster.wait = 0
```

Additional Information. [N/A]

2.83 DUMP 7007

Code	Symbol	Kernel Block(s)
7007	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: c_nodeStartMaster.failNr = 1
Node 2: c_nodeStartMaster.startInfoErrorCode = -202116109
Node 2: c_nodeStartMaster.blockLcp = 0, c_nodeStartMaster.blockGcp = 0
```

Additional Information. [N/A]

2.84 DUMP 7008

Code	Symbol	Kernel Block(s)
7008	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: cfirstDeadNode = -256, cstartPhase = 7, cnoReplicas = 2
Node 2: cwaitLcpSr = 0
```

Additional Information. [N/A]

2.85 DUMP 7009

Code	Symbol	Kernel Block(s)
7009	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: ccalcOldestRestorableGci = 1, cnoOfNodeGroups = 1
Node 2: cstartGcpNow = 0
Node 2: crestartGci = 1
```

Additional Information. [N/A]

2.86 DUMP 7010

Code	Symbol	Kernel Block(s)
7010	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: cminHotSpareNodes = 0, c_lcpState.lcpStatusUpdatedPlace = 9843, cLcpStart = 1
Node 2: c_blockCommit = 0, c_blockCommitNo = 0
```

Additional Information. [N/A]

2.87 DUMP 7011

Code	Symbol	Kernel Block(s)
7011	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: c_COPY_GCIREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_COPY_TABREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_CREATE_FRAGREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_DIH_SWITCH_REPLICA_REQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_EMPTY_LCP_REQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_END_TOREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_GCP_COMMIT_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_GCP_PREPARE_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_GCP_SAVEREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_INCL_NODEREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_MASTER_GCPREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_MASTER_LCPREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_START_INFOREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_START_RECREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_START_TOREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_STOP_ME_REQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_TC_CLOPSIZEREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_TCGETOPSIZEREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
Node 2: c_UPDATE_TOREQ_Counter = [SignalCounter: m_count=0 0000000000000000]
```

Additional Information. [N/A]

2.88 DUMP 7012

Code	Symbol	Kernel Block(s)
7012	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: ParticipatingDIH = 0000000000000000
Node 2: ParticipatingLQH = 0000000000000000
Node 2: m_LCP_COMPLETE_REP_Counter_DIH = [SignalCounter: m_count=0 0000000000000000]
Node 2: m_LCP_COMPLETE_REP_Counter_LQH = [SignalCounter: m_count=0 0000000000000000]
Node 2: m_LAST_LCP_FRAG_ORD = [SignalCounter: m_count=0 0000000000000000]
Node 2: m_LCP_COMPLETE_REP_From_Master_Received = 0
```

Additional Information. [N/A]

2.89 DUMP 7013

Code	Symbol	Kernel Block(s)
7013	DihDumpLCPState	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: lcpStatus = 0 (update place = 9843)
Node 2: lcpStart = 1 lcpStopGcp = 1 keepGci = 1 oldestRestorable = 1
Node 2: immediateLcpStart = 0 masterLcpNodeId = 2
```

Additional Information. [N/A]

2.90 DUMP 7014

Code	Symbol	Kernel Block(s)
7014	DihDumpLCPMasterTakeOver	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: c_lcpMasterTakeOverState.state = 0 updatePlace = 11756 failedNodeId = -202116109
Node 2: c_lcpMasterTakeOverState.minTableId = 4092851187 minFragId = 4092851187
```

Additional Information. [N/A]

2.91 DUMP 7015

Code	Symbol	Kernel Block(s)
7015	---	DBDIH

Description. Writes table fragment status output for NDB tables to the cluster log, in order of their table IDs. A starting table ID can optionally be specified, in which case tables having lower IDs than this are skipped; otherwise, status information for all NDB tables is included in the output.

Sample Invocation/Output. Invoking this command using the optional table ID argument gives the following output in the system shell:

```
shell> ndb_mgm -e 'ALL DUMP 2015 25'
Connected to Management Server at: localhost:1186
```

```
Sending dump signal with data:
0x00001b67 0x00000019
Sending dump signal with data:
0x00001b67 0x00000019
```

This causes Table 1 through Table 24 to be skipped in the output written into the cluster log, as shown here:

```
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5: Table 25: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5: Table 27: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5: Table 28: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=0(Idle) 1(on 6)=0
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5: Table 29: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=0(Idle) 1(on 6)=0
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 5:  Fragment 1: noLcpReplicas==0  0(on 6)=0(Idle) 1(on 5)=0
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6: Table 25: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6: Table 27: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6: Table 28: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 0: noLcpReplicas==0  0(on 5)=0(Idle) 1(on 6)=0
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6: Table 29: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 0: noLcpReplicas==0  0(on 5)=0(Idle) 1(on 6)=0
2016-07-21 13:08:29 [MgmtSrvr] INFO     -- Node 6:  Fragment 1: noLcpReplicas==0  0(on 6)=0(Idle) 1(on 5)=0
```

Additional Information. Output provided by `DUMP 7015` is the same as that provided by `DUMP 7021`, except that the latter includes only a single table specified by table ID. For more detailed information about the fields included in this output, see Section 2.97, “DUMP 7021”.

2.92 DUMP 7016

Code	Symbol	Kernel Block(s)
7016	`DihAllAllowNodeStart`	`DBDIH`

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.93 DUMP 7017

Code	Symbol	Kernel Block(s)
7017	`DihMinTimeBetweenLCP`	`DBDIH`

Description. [Unknown]

Sample Output.

```
```

```
...
```

Additional Information. [N/A]

2.94 DUMP 7018

Code	Symbol	Kernel Block(s)
7018	`DihMaxTimeBetweenLCP`	`DBDIH`

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.95 DUMP 7019

Code	Symbol	Kernel Block(s)
7019	---	`DBDIH`

Description. Write the distributed data block's view of node failure handling for a failed node (given its node ID) into the cluster log. Execute as `ALL DUMP 7019 FailedNodeId`.

Sample Output.

```
...
```

Additional Information. Added in NDB 7.2.9. (Bug #14220269)

2.96 DUMP 7020

Code	Symbol	Kernel Block(s)
7020	`---`	`DBDIH`

Description. This command provides general signal injection functionality. Two additional arguments are always required:

1. The number of the signal to be sent
2. The number of the block to which the signal should be sent

In addition some signals permit or require extra data to be sent.

Sample Output.

```
...
```

Additional Information. [N/A]

2.97 DUMP 7021

Code	Symbol	Kernel Block(s)
7021	`---`	DBDIH

Description. Writes table fragment status information for a single NDB table to the cluster log. DUMP 7015 is the same is this command, except that DUMP 7015 logs the information for multiple (or all) NDB tables.

The table to obtain information for is specified by table ID. You can find the ID for a given table in the output of ndb_show_tables, as shown here:

```
shell> ndb_show_tables
id    type                 state    logging database     schema   name
29    OrderedIndex         Online   No      sys          def      PRIMARY
1     IndexTrigger         Online   -                             NDB$INDEX_11_CUSTOM
3     IndexTrigger         Online   -                             NDB$INDEX_15_CUSTOM
8     UserTable            Online   Yes     mysql        def      NDB$BLOB_7_3
5     IndexTrigger         Online   -                             NDB$INDEX_28_CUSTOM
13    OrderedIndex         Online   No      sys          def      PRIMARY
10    UserTable            Online   Yes     test         def      n1
27    UserTable            Online   Yes     c            def      t1
...
```

Sample Invocation/Output. Using the table ID for table n1 found in the ndb_show_tables sample output shown previously (and highlighted therein), an invocation of this command might look like this when running ndb_mgm in the system shell:

```
shell> ndb_mgm -e 'ALL DUMP 7015 10'
Connected to Management Server at: localhost:1186
Sending dump signal with data:
0x00001b67 0x0000000a
Sending dump signal with data:
0x00001b67 0x0000000a
```

This writes the following output to the cluster log:

```
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 5: Table 10: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 5:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 6: Table 10: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 6:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 6:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 7: Table 10: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 7:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 7:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 8: Table 10: TabCopyStatus: 0 TabUpdateStatus: 0 TabLcpSta
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 8:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=
2016-07-21 12:12:11 [MgmtSrvr] INFO     -- Node 8:  Fragment 1: noLcpReplicas==0  0(on 6)=59(Idle) 1(on 5)=
```

Additional Information. More information about each of the fields written by DUMP 7021 into the cluster log is shown in the next few paragraphs. The enumerations are defined as properties of structure TabRecord in storage/ndb/src/kernel/blocks/dbdih/Dbdih.hpp.

TabCopyStatus (table copy status) takes one of the following values: 0: CS_IDLE, 1: CS_SR_PHASE1_READ_PAGES, 2: CS_SR_PHASE2_READ_TABLE, 3: CS_SR_PHASE3_COPY_TABLE, 4: CS_REMOVE_NODE, 5: CS_LCP_READ_TABLE, 6: CS_COPY_TAB_REQ, 7: CS_COPY_NODE_STATE,

8: CS_ADD_TABLE_MASTER, 9: CS_ADD_TABLE_SLAVE, 10: CS_INVALIDATE_NODE_LCP, 11: CS_ALTER_TABLE, 12: CS_COPY_TO_SAVE, 13: CS_GET_TABINFO.

TabUpdateStatus (table update status) takes one of the following values: 0: US_IDLE, 1: US_LOCAL_CHECKPOINT, 2: US_LOCAL_CHECKPOINT_QUEUED, 3: US_REMOVE_NODE, 4: US_COPY_TAB_REQ, 5: US_ADD_TABLE_MASTER, 6: US_ADD_TABLE_SLAVE, 7: US_INVALIDATE_NODE_LCP, 8: US_CALLBACK.

TabLcpStatus (table local checkpoint status) takes one of the following values: 1: TLS_ACTIVE, 2: TLS_WRITING_TO_FILE, 3: TLS_COMPLETED.

Table fragment information is also provided for each node. This is similar to what is shown here:

```
Node 5:  Fragment 0: noLcpReplicas==0  0(on 5)=59(Idle) 1(on 6)=59(Idle)
```

The node and fragment are identified by their IDs. noLcpReplicas represents the number of replicas remaining to be checkpointed by any ongoing LCP. The remainder of the line has the format shown here:

```
replica_id(on node_id)=lcp_id(status)
```

replica_id, *node_id*, and *lcp_id* are the IDs of, respectively, the replica, node, and local checkpoint. *status* is always one of Idle or Ongoing.

2.98 DUMP 7024

Code	Symbol	Kernel Block(s)
7024	---	DBDIH

Description. Determines whether tables are in their expected states.

Sample Output.

```
...
```

Additional Information. Added in NDB 7.2.17 and NDB 7.3.6. (Bug #18550318)

2.99 DUMP 7033

Code	Symbol	Kernel Block(s)
7033	DihFragmentsPerNode	DBDIH

Description. Prints the number of fragments on one or more data nodes. No arguments other than the node ID are used.

Sample Output. Output from ALL DUMP 7033 on an NDB Cluster with two data nodes and NoOfReplicas=2:

```
2014-10-13 19:07:44 [MgmtSrvr] INFO     -- Node 5: Fragments per node = 1
2014-10-13 19:07:44 [MgmtSrvr] INFO     -- Node 6: Fragments per node = 1
```

Additional Information. Added in NDB 7.4.1.

2.100 DUMP 7080

Code	Symbol	Kernel Block(s)
7080	EnableUndoDelayDataWrite	DBACC, DBDIH, DBTUP

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.101 DUMP 7090

Code	Symbol	Kernel Block(s)
7090	DihSetTimeBetweenGcp	DBDIH

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.102 DUMP 7098

Code	Symbol	Kernel Block(s)
7098	---	DBDIH

Description. [Unknown]

Sample Output.

```
Node 2: Invalid no of arguments to 7098 - startLcpRoundLoopLab - expected
2 (tableId, fragmentId)
```

Additional Information. [N/A]

2.103 DUMP 7099

Code	Symbol	Kernel Block(s)
7099	DihStartLcpImmediately	DBDIH

Description. Can be used to trigger an LCP manually.

Sample Output. In this example, node 2 is the master node and controls LCP/GCP synchronization for the cluster. Regardless of the *node_id* specified, only the master node responds:

```
Node 2: Local checkpoint 7 started. Keep GCI = 1003 oldest restorable GCI = 947
```

```
Node 2: Local checkpoint 7 completed
```

Additional Information. You may need to enable a higher logging level using the `CLUSTERLOG` `ndb_mgm` client command to have the checkpoint's completion reported, as shown here:

```
ndb_mgmgt; ALL CLUSTERLOG CHECKPOINT=8
```

2.104 DUMP 7901

Code	Symbol	Kernel Block(s)
7901	`---`	`DBDIH`, `DBLQH`

Description. Provides timings of GCPs.

Sample Output.

```
...
```

2.105 DUMP 8004

Code	Symbol	Kernel Block(s)
8004	`---`	`SUMA`

Description. Dumps information about subscription resources.

Sample Output.

```
Node 2: Suma: c_subscriberPool  size: 260 free: 258
Node 2: Suma: c_tablePool  size: 130 free: 128
Node 2: Suma: c_subscriptionPool  size: 130 free: 128
Node 2: Suma: c_syncPool  size: 2 free: 2
Node 2: Suma: c_dataBufferPool  size: 1009 free: 1005
Node 2: Suma: c_metaSubscribers count: 0
Node 2: Suma: c_removeDataSubscribers count: 0
```

Additional Information. When `subscriberPool ... free` becomes and stays very low relative to `subscriberPool ... size`, it is often a good idea to increase the value of the `MaxNoOfTables` configuration parameter (`subscriberPool` = 2 * `MaxNoOfTables`). However, there could also be a problem with API nodes not releasing resources correctly when they are shut down. `DUMP 8004` provides a way to monitor these values.

2.106 DUMP 8005

Code	Symbol	Kernel Block(s)
8005	`---`	`SUMA`

Description. [Unknown]

Sample Output.

```
Node 2: Bucket 0 10-0 switch gci: 0 max_acked_gci: 2961 max_gci: 0 tail: -256 head: -256
Node 2: Bucket 1 00-0 switch gci: 0 max_acked_gci: 2961 max_gci: 0 tail: -256 head: -256
```

Additional Information. [N/A]

2.107 DUMP 8010

Code	Symbol	Kernel Block(s)
8010	`---`	SUMA

Description. Writes information about all subscribers and connected nodes to the cluster log.

Sample Output. In this example, node 1 is a management node, nodes 2 and 3 are data nodes, and nodes 4 and 5 are SQL nodes (which both act as replication masters).

```
2010-10-15 10:08:33 [MgmtSrvr] INFO     -- Node 2: c_subscriber_nodes: 0000000000000000000000000000000000000000
2010-10-15 10:08:33 [MgmtSrvr] INFO     -- Node 2: c_connected_nodes: 0000000000000000000000000000000000000000
2010-10-15 10:08:33 [MgmtSrvr] INFO     -- Node 3: c_subscriber_nodes: 0000000000000000000000000000000000000000
2010-10-15 10:08:33 [MgmtSrvr] INFO     -- Node 3: c_connected_nodes: 0000000000000000000000000000000000000000
```

For each data node, this `DUMP` command prints two hexadecimal numbers. These are representations of bitfields having one bit per node ID, starting with node ID 0 for the rightmost bit (`0x01`).

The subscriber nodes bitmask (`c_subscriber_nodes`) has the significant hexadecimal digits `30` (decimal 48), or binary `110000`, which equates to nodes 4 and 5. The connected nodes bitmask (`c_connected_nodes`) has the significant hexadecimal digits `32` (decimal 50). The binary representation of this number is `110010`, which has `1` as the second, fifth, and sixth digits (counting from the right), and so works out to nodes 1, 4, and 5 as the connected nodes.

2.108 DUMP 8011

Code	Symbol	Kernel Block(s)
8011	`---`	SUMA

Description. Writes information to the cluster log about all subscribers in the cluster. When using this information, you should keep in mind that a table may have many subscriptions, and a subscription may have more than one subscriber. The output from `DUMP 8011` includes the following information:

- *For each table*: The table ID, version number, and total number of subscribers
- *For each subscription to a given table*: The subscription ID
- *For each subscriber belonging to a given subscription*: The subscriber ID, sender reference, sender data, and subscription ID

Sample Output. (From cluster log:)

```
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1: -- Starting dump of subscribers --
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1: Table: 2 ver: 4294967040 #n: 1 (ref,data,sub
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1:  [ 80010004 24 0 ]
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1: Table: 3 ver: 4294967040 #n: 1 (ref,data,sub
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1:  [ 80010004 28 1 ]
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1: Table: 4 ver: 4294967040 #n: 1 (ref,data,sub
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1:  [ 80020004 24 2 ]
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 1: -- Ending dump of subscribers --
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 2: -- Starting dump of subscribers --
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 2: Table: 2 ver: 4294967040 #n: 1 (ref,data,sub
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 2:  [ 80010004 24 0 ]
Sunday 01 November 2015 13:17:31 [MgmtSrvr] INFO     -- Node 2: Table: 3 ver: 4294967040 #n: 1 (ref,data,sub
```

```
Sunday 01 November 2015 13:17:31 [MgmSrvr] INFO     -- Node 2:  [ 80010004 28 1 ]
Sunday 01 November 2015 13:17:31 [MgmSrvr] INFO     -- Node 2: Table: 4 ver: 4294967040 #n: 1 (ref,data,subscr
Sunday 01 November 2015 13:17:31 [MgmSrvr] INFO     -- Node 2:  [ 80020004 24 2 ]
Sunday 01 November 2015 13:17:31 [MgmSrvr] INFO     -- Node 2: -- Ending dump of subscribers --
```

2.109 DUMP 8013

Code	Symbol	Kernel Block(s)
8013	`---`	SUMA

Description. Writes a dump of all lagging subscribers to the cluster log.

Sample Output. This example shows what is written to the cluster log after `ALL DUMP 8013` is executed on a 4-node cluster:

```
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 5: -- Starting dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 5: Highest epoch 1632087572485, oldest epoch 1632087572485
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 5: -- End dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 5: Reenable event buffer
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 6: -- Starting dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 6: Highest epoch 1632087572486, oldest epoch 1632087572486
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 6: -- End dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 7: -- Starting dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 7: Highest epoch 1632087572486, oldest epoch 1632087572486
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 7: -- End dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 8: -- Starting dump of pending subscribers --
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 8: Highest epoch 1632087572486, oldest epoch 1632087572486
2013-05-27 13:59:02 [MgmtSrvr] INFO     -- Node 8: -- End dump of pending subscribers --
```

Additional Information. Added in NDB 7.2.13. (Bug #16203623)

2.110 DUMP 9002

Code	Symbol	Kernel Block(s)
9002	DumpTsman	TSMAN

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.111 DUMP 9800

Code	Symbol	Kernel Block(s)
9800	DumpTsman	TSMAN

Description. Kills data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 18:32:53
```

```
Status: Temporary error, restart node
Message: Internal program error (failed ndbrequire) (Internal error,
programming error or missing error message, please report a bug)
Error: 2341
Error data: tsman.cpp
Error object: TSMAN (Line: 1413) 0x0000000a
Program: ./libexec/ndbd
Pid: 29658
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.112 DUMP 9801

Code	Symbol	Kernel Block(s)
9801	---	TSMAN

Description. Kills data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 18:35:48
Status: Temporary error, restart node
Message: Internal program error (failed ndbrequire) (Internal error,
programming error or missing error message, please report a bug)
Error: 2341
Error data: tsman.cpp
Error object: TSMAN (Line: 1844) 0x0000000a
Program: ./libexec/ndbd
Pid: 30251
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.113 DUMP 9802

Code	Symbol	Kernel Block(s)
9802	---	TSMAN

Description. Kills data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 18:39:30
Status: Temporary error, restart node
Message: Internal program error (failed ndbrequire) (Internal error,
programming error or missing error message, please report a bug)
Error: 2341
Error data: tsman.cpp
Error object: TSMAN (Line: 1413) 0x0000000a
Program: ./libexec/ndbd
Pid: 30482
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.114 DUMP 9803

Code	Symbol	Kernel Block(s)
9803	---	TSMAN

Description. Kills data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 18:41:32
Status: Temporary error, restart node
Message: Internal program error (failed ndbrequire) (Internal error,
programming error or missing error message, please report a bug)
Error: 2341
Error data: tsman.cpp
Error object: TSMAN (Line: 2144) 0x0000000a
Program: ./libexec/ndbd
Pid: 30712
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

2.115 DUMP 10000

Code	Symbol	Kernel Block(s)
10000	DumpLgman	---

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.116 DUMP 11000

Code	Symbol	Kernel Block(s)
11000	DumpPgman	---

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.117 DUMP 12001

Code	Symbol	Kernel Block(s)
12001	TuxLogToFile	DBTUX

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.118 DUMP 12002

Code	Symbol	Kernel Block(s)
12002	TuxSetLogFlags	DBTUX

Description. [Unknown]

Sample Output.

```
...
```

Additional Information. [N/A]

2.119 DUMP 12009

Code	Symbol	Kernel Block(s)
12009	TuxMetaDataJunk	DBTUX

Description. Kills data node.

Sample Output.

```
Time: Sunday 01 November 2015 - 19:49:59
Status: Temporary error, restart node
Message: Error OS signal received (Internal error, programming error or
missing error message, please report a bug)
Error: 6000
Error data: Signal 6 received; Aborted
Error object: main.cpp
Program: ./libexec/ndbd
Pid: 13784
Trace: /usr/local/mysql/cluster/ndb_2_trace.log.1
Version: Version 5.6.27-ndb-7.4.8
```

Additional Information. [N/A]

Chapter 3 The NDB Communication Protocol

Table of Contents

3.1 NDB Protocol Overview 65
3.2 NDB Protocol Messages 66
3.3 Operations and Signals 66

This chapter provides information about the protocol used for communication between data nodes and API nodes in an NDB Cluster to perform various operations such as data reads and writes, committing and rolling back transactions, and handling of transaction records.

3.1 NDB Protocol Overview

NDB Cluster data and API nodes communicate with one another by passing messages to one another. The sending of a message from one node and its reception by another node is referred to as a *signal*; the `NDB` Protocol is the set of rules governing the format of these messages and the manner in which they are passed.

An `NDB` message is typically either a *request* or a *response*. A request indicates that an API node wants to perform an operation involving cluster data (such as retrieval, insertion, updating, or deletion) or transactions (commit, roll back, or to fetch or relase a transaction record). A request is, when necessary, accompanied by key or index information. The response sent by a data node to this request indicates whether or not the request succeeded and, where appropriate, is accompanied by one or more data messages.

Request types. A request is represented as a `REQ` message. Requests can be divided into those handling data and those handling transactions:

- **Data requests.** Data request operations are of three principal types:

  1. *Primary key lookup operations* are performed through the exchange of `TCKEY` messages.
  2. *Unique key lookup operations* are performed through the exchange of `TCINDX` messages.
  3. *Table or index scan operations* are performed through the exchange of `SCANTAB` messages.

  Data request messages are often accompanied by `KEYINFO` messages, `ATTRINFO` messages, or both sorts of messages.

- **Transactional requests.** These may be divided into two categories:

  1. *Commits and rollbacks*, which are represented by `TC_COMMIT` and `TCROLLBACK` request messages, respectively.
  2. *Transaction record requests*, consisting of transaction record acquisition and release, are handled through the use of, respectively, `TCSEIZE` and `TCRELEASE` request messages.

Response types. A response indicates either the success or the failure of the request to which it is sent in reply:

- A response indicating success is represented as a `CONF` (confirmation) message, and is often accompanied by data, which is packaged as one or more `TRANSID_AI` messages.

- A response indicating failure is represented as a REF (refusal) message.

For more information about these message types and their relationship to one another, see Section 3.2, “NDB Protocol Messages”.

3.2 NDB Protocol Messages

This section describes the NDB Protocol message types, their function, and their structure.

Naming Conventions. Message names are constructed according to a simple pattern which should be readily apparent from the discussion of request and response types in the previous section. These are shown in the following matrix:

Operation Type	Request (REQ)	Response/Success (CONF)	Response/Failure (REF)
Primary Key Lookup (TCKEY)	TCKEYREQ	TCKEYCONF	TCKEYREF
Unique Key Lookup (TCINDX)	TCINDXREQ	TCINDXCONF	TCINDXREF
Table or Index Scan (SCANTAB)	SCANTABREQ	SCANTABCONF	SCANTABREF
Result Retrieval (SCAN_NEXT)	SCAN_NEXTREQ	SCANTABCONF	SCANTABREF
Transaction Record Acquisition (TCSEIZE)	TCSEIZEREQ	TCSEIZECONF	TCSEIZEREF
Transaction Record Release (TCRELEASE)	TCRELEASEREQ	TCRELEASECONF	TCRELEASEREF

CONF and REF are shorthand for “confirmed” and “refused”, respectively.

Three additional types of messages are used in some instances of inter-node communication. These message types are listed here:

1. A KEYINFO message contains information about the key used in a TCKEYREQ or TCINDXREQ message. It is employed when the key data does not fit within the request message. KEYINFO messages are also sent for index scan operations in which bounds are employed.

2. An ATTRINFO message contains nonkey attribute values which does not fit within a TCKEYREQ, TCINDXREQ, or SCANTABREQ message. It is used for:

   - Supplying attribute values for inserts and updates
   - Designating which attributes are to be read for read operations
   - Specifying optional values to read for delete operations

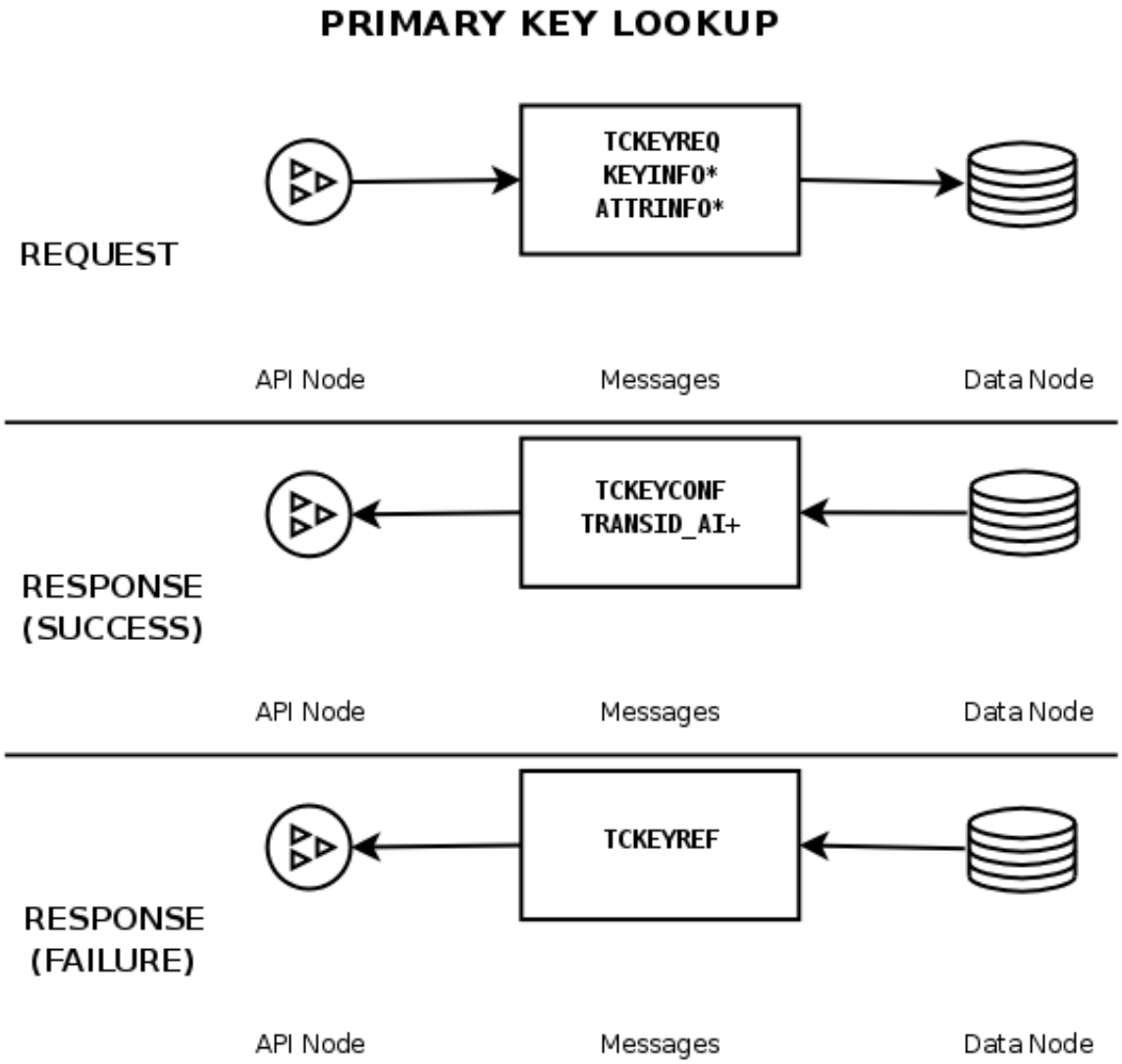
3. A TRANSID_AI message contains data returned from a read operation; in other words, it is a result set (or part of one).

3.3 Operations and Signals

In this section we discuss the sequence of message-passing that takes place between a data node and an API node for each of the following operations:

- Primary key lookup
- Unique key lookup
- Table scan or index scan
- Explicit commit of a transaction
- Rollback of a transaction
- Transaction record handling (acquisition and release)

Primary key lookup. An operation using a primary key lookup is performed as shown in the following diagram:



Note

***** and **+** are used here with the meanings “zero or more” and “one or more”, respectively.

The steps making up this process are listed and explained in greater detail here:

1. The API node sends a TCKEYREQ message to the data node. In the event that the necessary information about the key to be used is too large to be contained in the TCKEYREQ, the message may be accompanied by any number of KEYINFO messages carrying the remaining key information. If additional attributes are used for the operation and exceed the space available in the TCKEYREQ, or if data is to be sent to the data node as part of a write operation, then these are sent with the TCKEYREQ as any number of ATTRINFO messages.

2. The data node sends a message in response to the request, according to whether the operation succeeded or failed:

   - If the operation was successful, the data node sends a TCKEYCONF message to the API node. If the request was for a read operation, then TCKEYCONF is accompanied by a TRANSID_AI message, which contains actual result data. If there is more data than can be contained in a single TRANSID_AI can carry, more than one of these messages may be sent.

   - If the operation failed, then the data node sends a TCKEYREF message back to the API node, and no more signalling takes place until the API node makes a new request.

Unique key lookup. This is performed in a manner similar to that performed in a primary key lookup:

1. A request is made by the API node using a TCINDXREQ message which may be accompanied by zero or more KEYINFO messages, zero or more ATTRINFO messages, or both.

2. The data node returns a response, depending on whether or not the operation succeeded:

   - If the operation was a success, the message is TCINDXCONF. For a successful read operation, this message may be accompanied by one or more TRANSID_AI messages carrying the result data.

   - If the operation failed, the data node returns a TCINDXREF message.

The exchange of messages involved in a unique key lookup is illustrated in the following diagram:

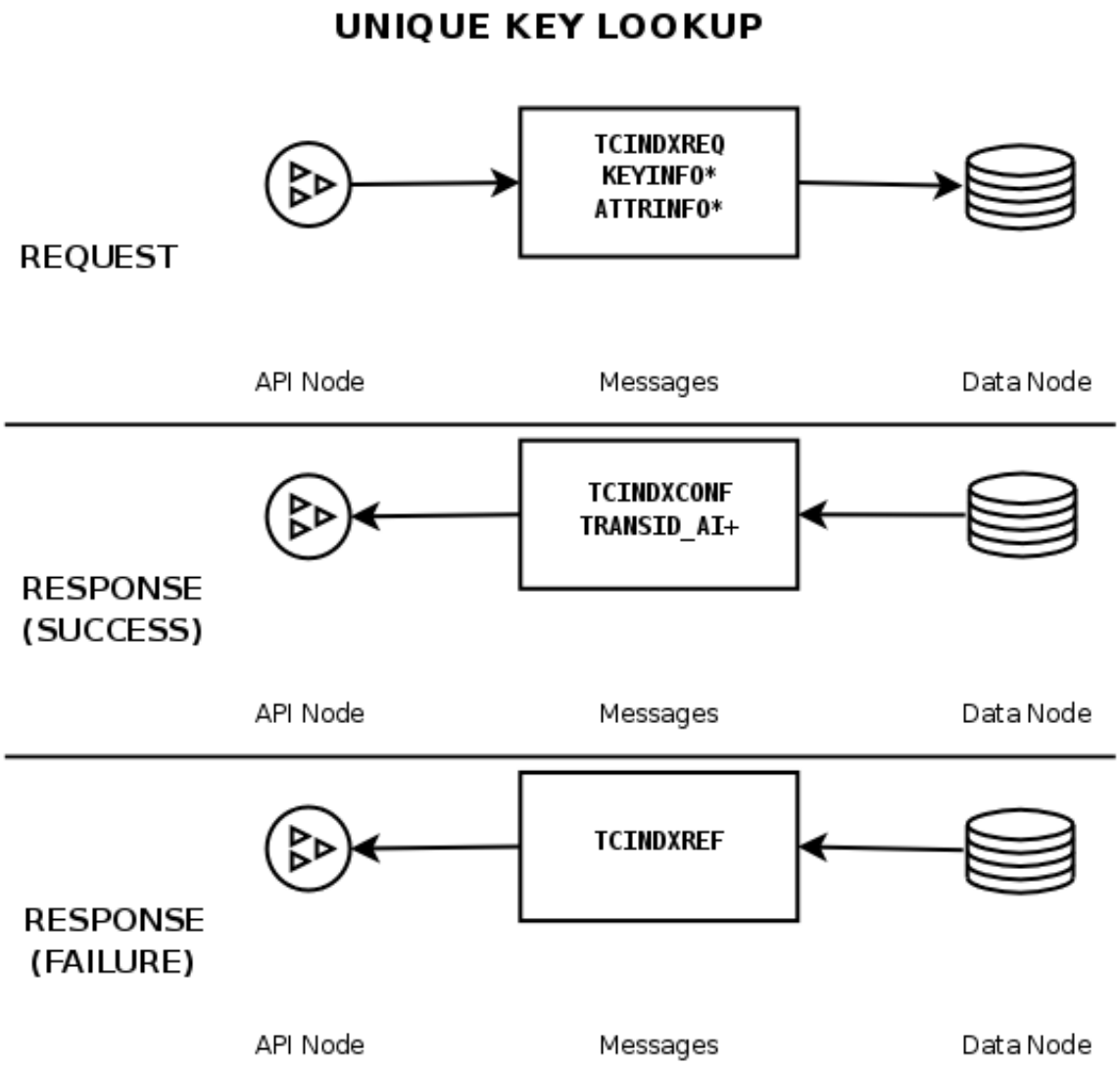
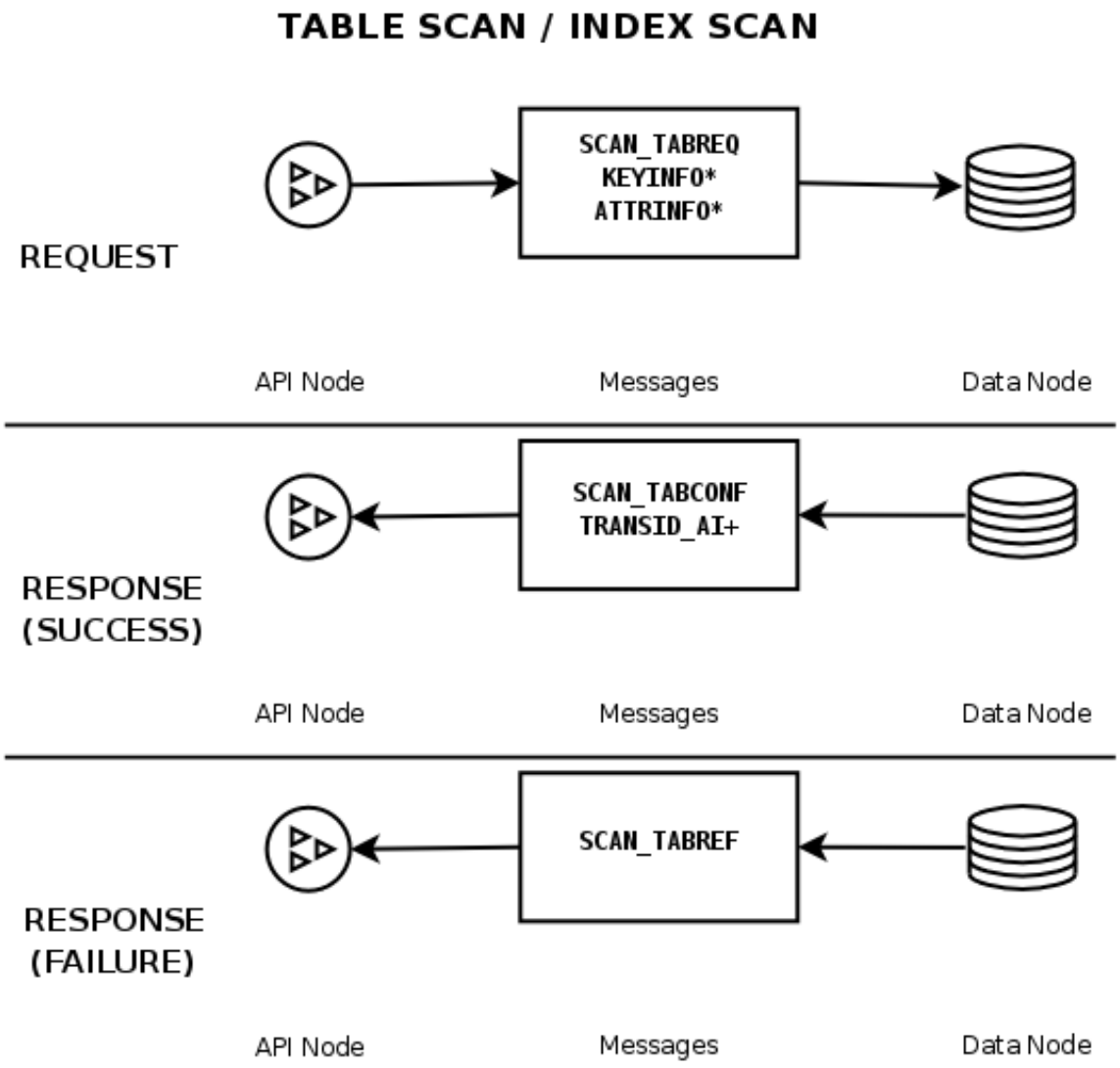


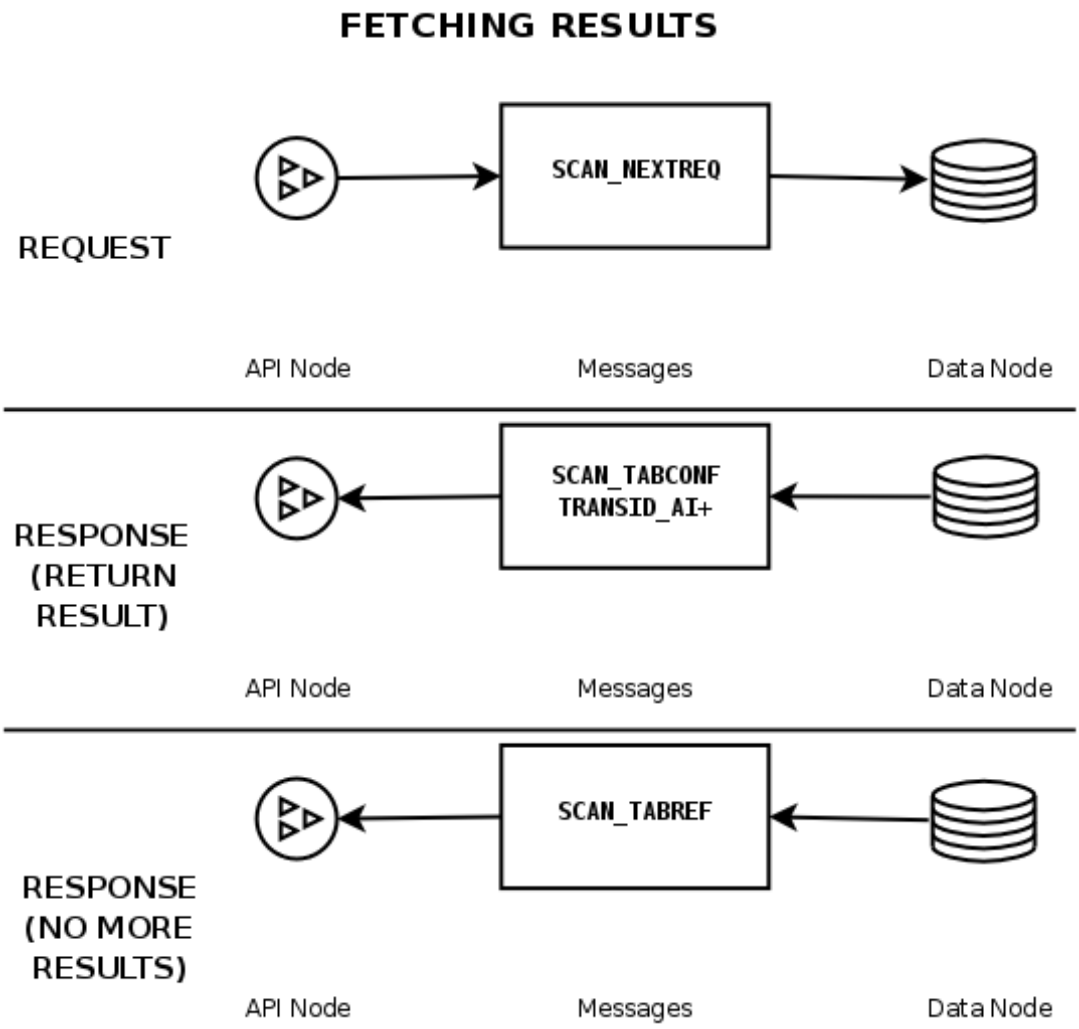
Table scans and index scans. These are similar in many respects to primary key and unique key lookups, as shown here:



1. A request is made by the API node using a `SCAN_TABREQ` message, along with zero or more `ATTRINFO` messages. `KEYINFO` messages are also used with index scans in the event that bounds are used.

2. The data node returns a response, according to whether or not the operation succeeded:

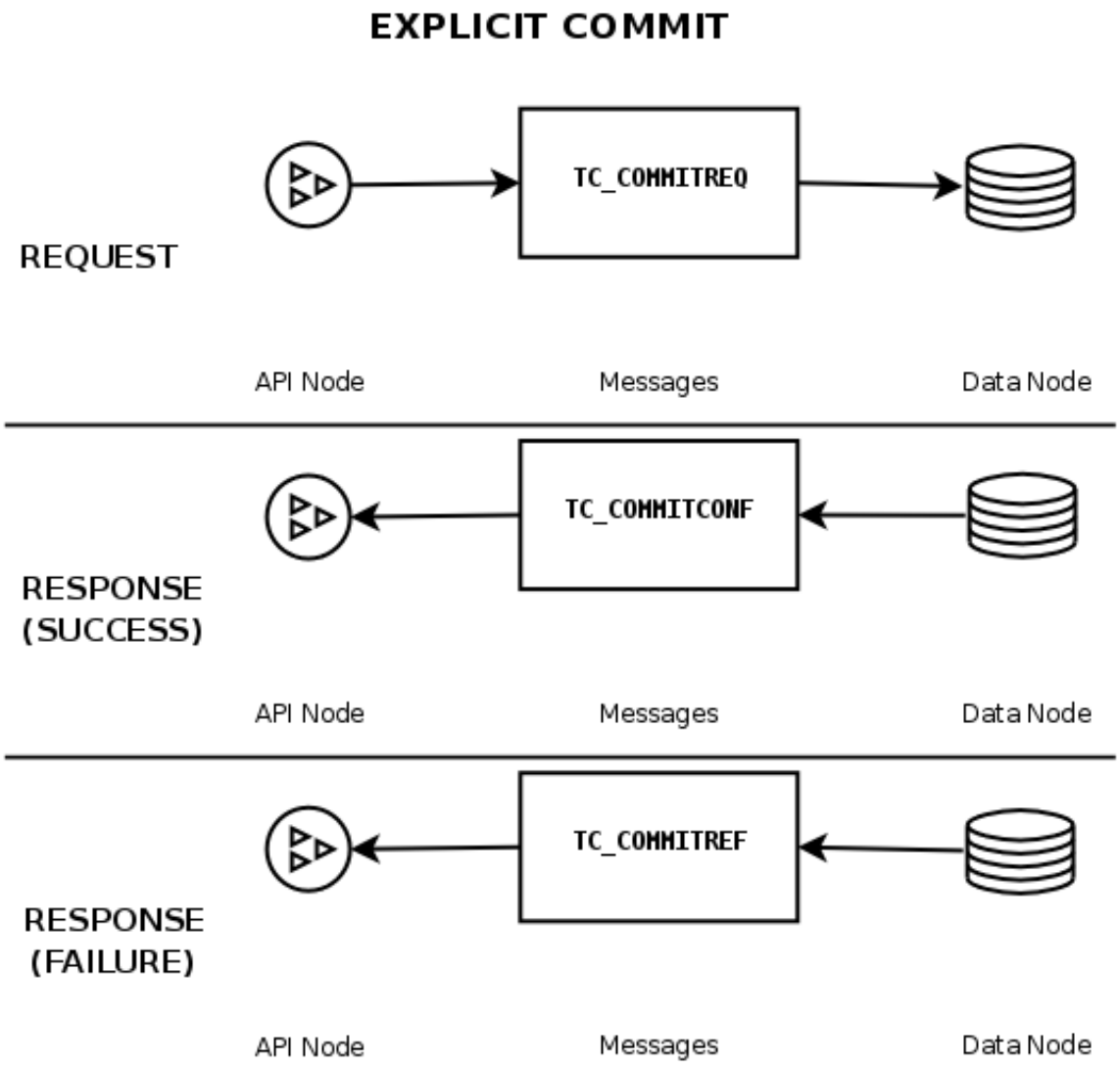
   - If the operation was a success, the message is `SCAN_TABCONF`. For a successful read operation, this message may be accompanied by one or more `TRANSID_AI` messages carrying the result data. However—unlike the case with lookups based on a primary or unique key—it is often necessary to fetch multiple results from the data node. Requests following the first are signalled by the API node using a `SCAN_NEXTREQ`, which tells the data node to send the next set of results (if there are more results). This is shown here:



- If the operation failed, the data node returns a `SCAN_TABREF` message.

  `SCAN_TABREF` is also used to signal to the API node that all data resulting from a read has been sent.

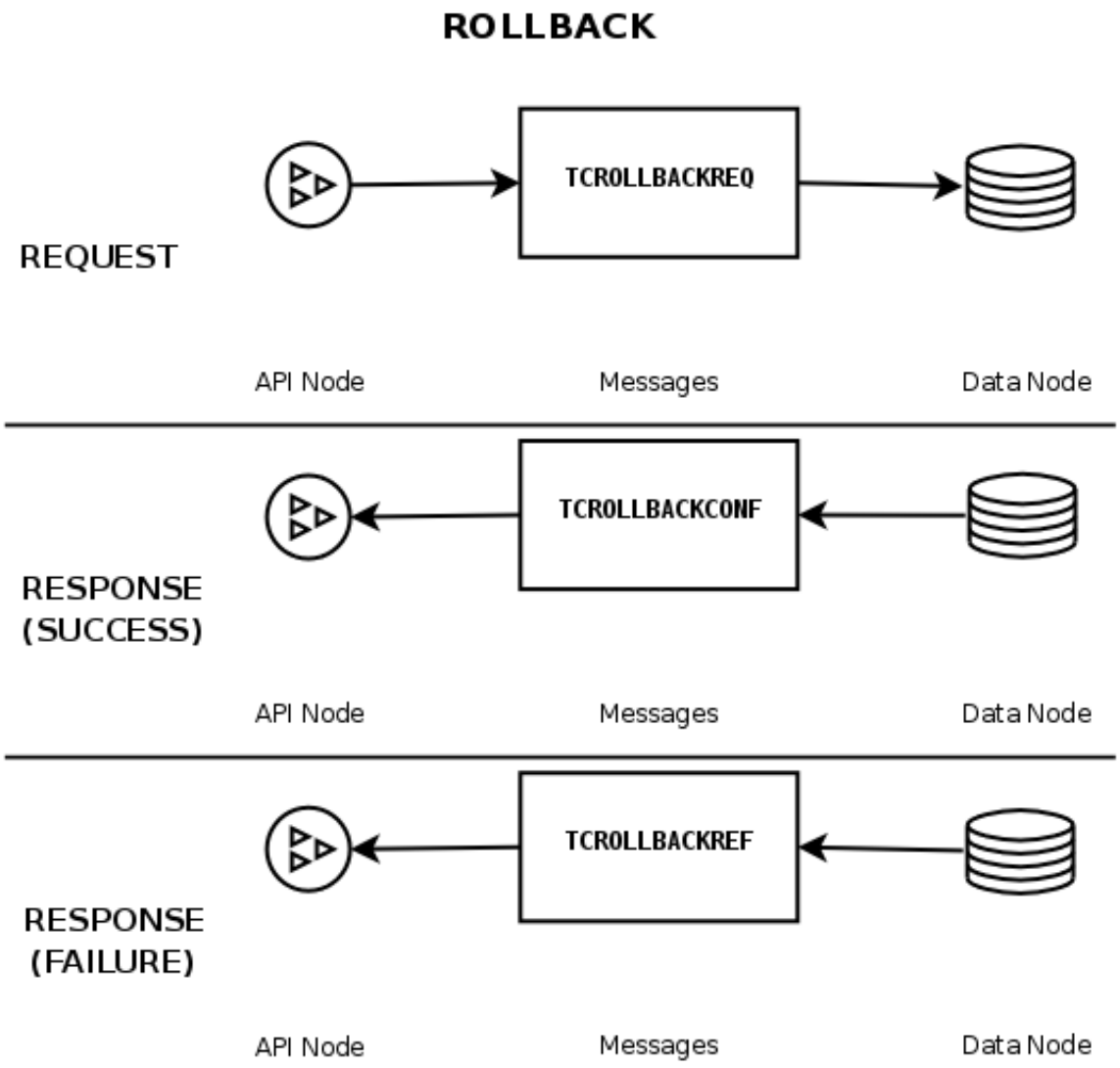
Committing and rolling back transactions. The process of performing an explicit commit follows the same general pattern as shown previously. The API node sends a `TC_COMMITREQ` message to the data node, which responds with either a `TC_COMMITCONF` (on success) or a `TC_COMMITREF` (if the commit failed). This is shown in the following diagram:



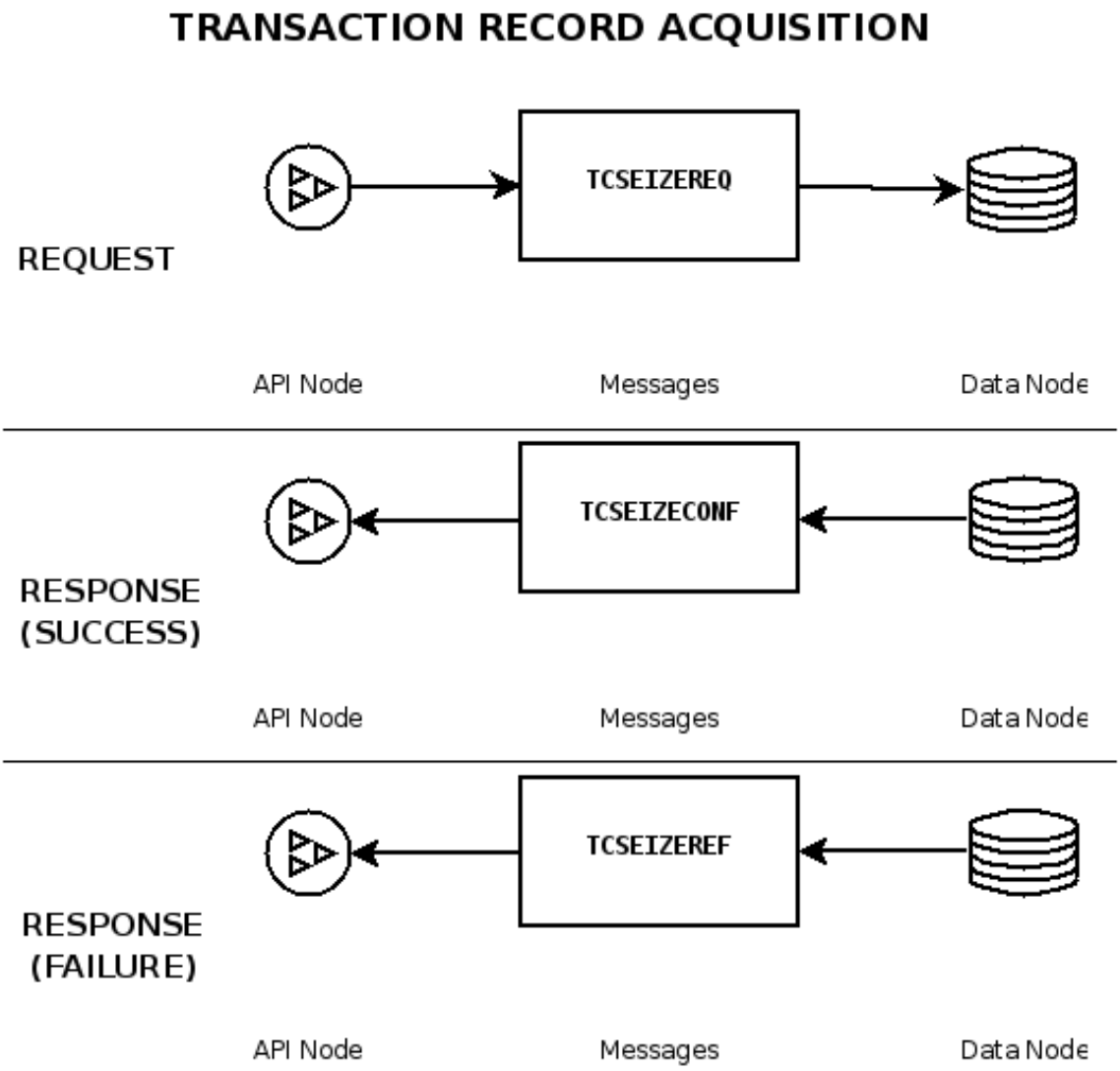
Note

Some operations perform a `COMMIT` automatically, so this is not required for every transaction.

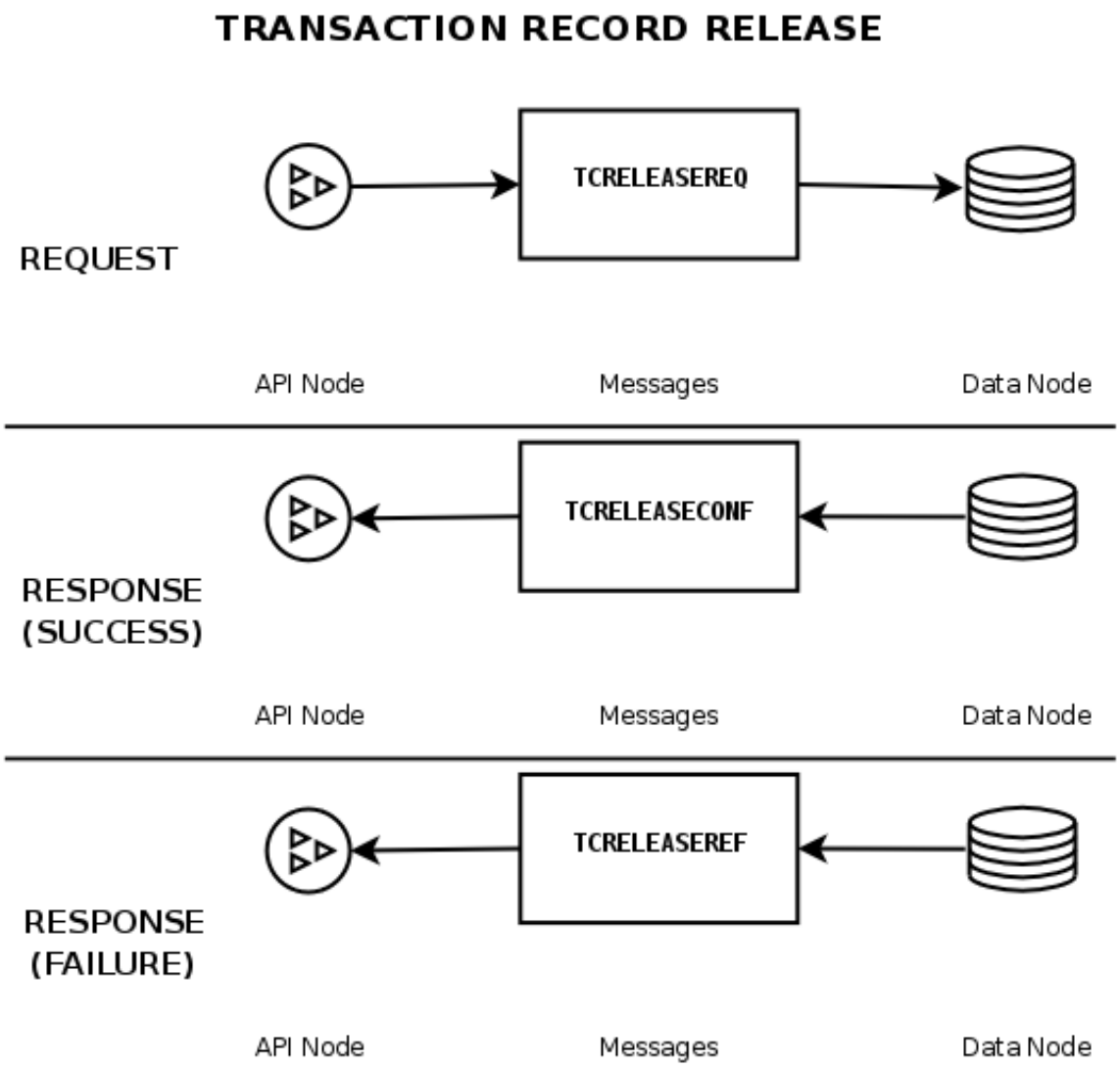
Rolling back a transaction also follows this pattern. In this case, however, the API node sends a `TCROLLBACKTREQ` message to the data node. Either a `TCROLLACKCONF` or a `TCROLLBACKREF` is sent in response, as shown here:



Handling of transaction records. Acquiring a transaction record is accomplished when an API node transmits a `TCSEIZEREQ` message to a data node and receives a `TCSEIZECONF` or `TCSEIZEREF` in return, depending on whether or not the request was successful. This is depicted here:



The release of a transaction record is also handled using the request-response pattern. In this case, the API node's request contains a `TCRELEASEREQ` message, and the data node's response uses either a `TCRELEASECONF` (indicating that the record was released) or a `TCRELEASEREF` (indicating that the attempt at release did not succeed). This series of events is illustrated in the next diagram:


TRANSACTION RECORD RELEASE
REQUEST
TCRELEASEREQ
API Node
Messages
Data Node
RESPONSE
(SUCCESS)
TCRELEASECONF
API Node
Messages
Data Node
RESPONSE
(FAILURE)
TCRELEASEREF
API Node
Messages
Data Node

Chapter 4 NDB Kernel Blocks

Table of Contents

4.1 The BACKUP Block 78
4.2 The CMVMI Block 78
4.3 The DBACC Block 78
4.4 The DBDICT Block 79
4.5 The DBDIH Block 79
4.6 The DBINFO Block 80
4.7 The DBLQH Block 80
4.8 The DBSPJ Block 82
4.9 The DBTC Block 82
4.10 The DBTUP Block 84
4.11 The DBTUX Block 85
4.12 The DBUTIL Block 86
4.13 The LGMAN Block 86
4.14 The NDBCNTR Block 86
4.15 The NDBFS Block 86
4.16 The PGMAN Block 87
4.17 The QMGR Block 87
4.18 The RESTORE Block 88
4.19 The SUMA Block 88
4.20 The THRMAN Block 88
4.21 The TRPMAN Block 89
4.22 The TSMAN Block 89
4.23 The TRIX Block 89

This chapter provides information about the major software modules making up the `NDB` kernel. The files containing the implementations of these blocks can be found in several directories under `storage/ndb/src/kernel/blocks/` in the NDB Cluster source tree.

As described elsewhere, the `NDB` kernel makes use of a number of different threads to perform various tasks. Kernel blocks are associated with these threads as shown in the following table:

Table 4.1 NDB Kernel Blocks and Threads

Thread (`ThreadConfig` Name)	**Kernel Blocks**
Main (`main`)	`CMVMI` (primary), `DBINFO`, `DBDICT`, `DBDIH`, `NDBCNTR`, `QMGR`, `DBUTIL`
LDM (`ldm`)	`DBTUP`, `DBACC`, `DBLQH` (primary), `DBTUX`, `BACKUP`, `TSMAN`, `LGMAN`, `PGMAN`, `RESTORE`
TC (`tc`)	`DBTC` (primary), `TRIX`
Replication (`rep`)	`SUMA` (primary), `DBSPJ`
Receiver (`recv`)	`CMVMI`
Sender (`send`)	`CMVMI`
I/O (`io`)	`NDBFS`

You can obtain more information about these threads from the documentation for the `ThreadConfig` data node configuration parameter.

4.1 The BACKUP Block

This block is responsible for handling online backups and checkpoints. It is found in `storage/ndb/src/kernel/blocks/backup/`, and contains the following files:

- `Backup.cpp`: Defines methods for node signal handling; also provides output methods for backup status messages to user.
- `BackupFormat.hpp`: Defines the formats used for backup data, `.CTL`, and log files.
- `Backup.hpp`: Defines the `Backup` class.
- `BackupInit.cpp`: Actual `Backup` class constructor is found here.
- `Backup.txt`: Contains a backup signal diagram (text format). Somewhat dated (from 2003), but still potentially useful for understanding the sequence of events that is followed during backups.
- `FsBuffer.hpp`: Defines the `FsBuffer` class, which implements the circular data buffer that is used (together with the NDB file system) for reading and writing backup data and logs.
- `read.cpp`: Contains some utility functions for reading log and checkpoint files to `STDOUT`.

4.2 The CMVMI Block

This block is responsible for configuration management between the kernel blocks and the `NDB` virtual machine, as well as the cluster job que and cluster transporters. It is found in `storage/ndb/src/kernel/blocks/cmvmi`, and contains these files:

- `Cmvmi.cpp`: Implements communication and reporting methods for the `Cmvmi` class.
- `Cmvmi.hpp`: Defines the `Cmvmi` class.

During startup, this block allocates and touches most of the memory needed for buffers used by the `NDB` kernel, such as those defined by `IndexMemory`, `DataMemory`, and `DiskPageBufferMemory`. At that time, `CMVMI` also gets the starting order of the nodes, and performs a number of functions whereby software modules can affect the runtime environment.

4.3 The DBACC Block

Also referred to as the `ACC` block, this is the access control and lock management module. It is also responsible for storing primary key and unique key hash indexes are stored. This block is found in `storage/ndb/src/kernel/blocks/dbacc`, which contains the following files:

- `Dbacc.hpp`: Defines the `Dbacc` class, along with structures for operation, scan, table, and other records.
- `DbaccInit.cpp`: `Dbacc` class constructor and destructor; methods for initialising data and records.
- `DbaccMain.cpp`: Implements `Dbacc` class methods.

The `ACC` block handles database index structures, which are stored in 8K pages. Database locks are also handled in the `ACC` block.

When a new tuple is inserted, the `TUP` block stores the tuple in a suitable space and returns an index (a reference to the address of the tuple in memory). `ACC` stores both the primary key and this tuple index of the tuple in a hash table.

Like the TUP block, the ACC block implements part of the checkpoint protocol. It also performs undo logging. It is implemented by the Dbacc class, which is defined in storage/ndb/src/kernel/blocks/dbacc/DbaccMain.hpp.

See also Section 4.10, “The DBTUP Block”.

4.4 The DBDICT Block

This block, the data dictionary block, is found in storage/ndb/src/kernel/blocks/dbdict. Data dictionary information is replicated to all DICT blocks in the cluster. This is the only block other than DBTC to which applications can send direct requests.

DBDICT is responsible for managing metadata (using the master data node) including the definitions for tables, columns, indexes, tablespaces, log files, log file groups, and other data objects.

This block is implemented in the following files:

- CreateIndex.txt: Contains notes about processes for creating, altering, and dropping indexes and triggers.
- Dbdict.cpp: Implements structure for event metadata records (for NDB$EVENTS_0), as well as methods for system start and restart, table and schema file handling, and packing table data into pages. Functionality for determining node status and handling node failures is also found here. In addition, this file implements data and other initialisation routines for Dbdict.
- DictLock.txt: Implementation notes: Describes locking of the master node's DICT against schema operations.
- printSchemaFile.cpp: Contains the source for the ndb_print_schema_file utility.
- Slave_AddTable.sfl: A signal log trace of a table creation operation for DBDICT on a nonmaster node.
- CreateTable.txt: Notes outlining the table creation process (dated).
- CreateTable.new.txt: Notes outlining the table creation process (updated version of CreateTable.txt).
- Dbdict.hpp: Defines the Dbdict class; also creates the NDB$EVENTS_0 table. Also defines a number of structures such as table and index records, as well as for table records.
- DropTable.txt: Implementation notes for the process of dropping a table.
- Dbdict.txt: Implementation notes for creating and dropping events and NdbEventOperation objects.
- Event.txt: A copy of Dbdict.txt.
- Master_AddTable.sfl: A signal log trace of a table creation operation for DBDICT on the master node.
- SchemaFile.hpp: Defines the structure of a schema file.

4.5 The DBDIH Block

This block provides data distribution management services for distribution information about each table, table partition, and replica of each partition. It is also responsible for handling of local and global checkpoints. DBDIH also manages node and system restarts. This block is implemented in the following files, all found in the directory storage/ndb/src/kernel/blocks/dbdih:

- `Dbdih.hpp`: This file contains the definition of the `Dbdih` class, as well as the `FileRecordPtr` type, which is used to keep storage information about a fragment and its replicas. If a fragment has more than one backup replica, then a list of the additional ones is attached to this record. This record also stores the status of the fragment, and is 64-byte aligned.
- `DbdihMain.cpp`: Contains definitions of `Dbdih` class methods.
- `printSysfile/printSysfile.cpp`: Older version of the `printSysfile.cpp` in the main `dbdih` directory.
- `DbdihInit.cpp`: Initializes `Dbdih` data and records; also contains the class destructor.
- `LCP.txt`: Contains developer notes about the exchange of messages between `DIH` and `LQH` that takes place during a local checkpoint.
- `printSysfile.cpp`: This file contains the source code for `ndb_print_sys_file`. For information about using this utility, see `ndb_print_sys_file` — Print NDB System File Contents.
- `Sysfile.hpp`: Contains the definition of the `Sysfile` structure; in other words, the format of an `NDB` system file. See Chapter 1, *NDB Cluster File Systems*, for more information about `NDB` system files.

This block often makes use of `BACKUP` blocks on the data nodes to accomplish distributed tasks, such as global checkpoints and system restarts.

This block is implemented as the `Dbdih` class, whose definition may be found in the file `storage/ndb/src/kernel/blocks/dbdih/Dbdih.hpp`.

4.6 The DBINFO Block

The `DBINFO` block provides support for the `ndbinfo` information database used to obtain information about data node internals.

An API node communicates with this block to retrieve `ndbinfo` data using `DBINFO_SCANREQ` and `DBINFO_SCANCONF` signals. The API node communicates with `DBINFO` on the master data node, which communicates with `DBINFO` on the remaining data nodes. The `DBINFO` block on each data node fetches information from the other kernel blocks on that node, including `DBACC`, `DBTUP`, `BACKUP`, `DBTC`, `SUMA`, `DBUTIL`, `TRIX`, `DBTUX`, `DBDICT`, `CMVMI`, `DBLQH`, `LGMAN`, `PGMAN`, `DBSPJ`, `THRMAN`, `TRPMAN`, and `QMGR`. The local `DBINFO` then sends the information back to `DBINFO` on the master node, which in turn passes it back to the API node.

This block is implemented in the file `storage/ndb/src/kernel/blocks/dbinfo/Dbinfo.hpp` as the `Dbinfo` class. The file `Dbinfo.cpp` in the same directory defines the methods of this class (mostly signal handlers). Also in the dbinfo directory is a text file `DbinfoScan.txt` which provides information about `DBINFO` messaging.

4.7 The DBLQH Block

This is the local, low-level query handler block, which manages data and transactions local to the cluster's data nodes, and acts as a coordinator of 2-phase commits. It is responsible (when called on by the transaction coordinator) for performing operations on tuples, accomplishing this task with help of `DBACC` block (which manages the index structures) and `DBTUP` (which manages the tuples). It is made up of the following files, found in `storage/ndb/src/kernel/blocks/dblqh`:

- `Dblqh.hpp`: Contains the `Dblqh` class definition. The code itself includes the following modules:
  - **Start/Restart Module.** This module handles the following start phases:
    - **Start phase 1.** Load block reference and processor ID

  - **Start phase 2.** Initiate all records within the block; connect LQH with ACC and TUP
  - **Start phase 4.** Connect each LQH with every other LQH in the database system. For an initial start, create the fragment log files. For a system restart or node restart, open the fragment log files and find the end of the log files.

- **Fragment addition and deletion module.** Used by the data dictionary to create new fragments and delete old fragments.
- **Execution module.** This module handles the reception of LQHKEYREQ messages and all processing of operations on behalf of this request. This also involves reception of various types of ATTRINFO and KEYINFO messages, as well as communications with ACC and TUP.
- **Log module.** The log module handles the reading and writing of the log. It is also responsible for handling system restarts, and controls system restart in TUP and ACC as well.
- **Tramsaction module.** This module handles the commit and completion phases.
- **TC failure module.** Handles failures in the transaction coordinator.
- **Scan module.** This module contains the code that handles a scan of a particular fragment. It operates under the control of the transaction coordinator and orders ACC to perform a scan of all tuples in the fragment. TUP performs the necessary search conditions to insure that only valid tuples are returned to the application.
- **Node recovery module.** This is used when a node has failed, copying the effected fragment to a new fragment replica. It also shuts down all connections to the failed node.
- **LCP module.** This module handles execution and control of local checkpoints in TUP and ACC. It also interacts with DIH to determine which global checkpoints are recoverable.
- **Global checkpoint module.** Assists DIH in discovering when GCPs are recoverable, and handles the GCP_SAVEREQ message requesting that LQH save a given GCP to disk and provide a notification of when this has been done.
- **File handling module.** This includes a number of sub-modules:
  - Signal reception
  - Normal operation
  - File change
  - Initial start
  - System restart, Phase 1
  - System restart, Phase 2
  - System restart, Phase 3
  - System restart, Phase 4
  - Error

- `DblqhInit.cpp`: Initialises `Dblqh` records and data. Also includes the `Dblqh` class destructor, used for deallocating these.
- `DblqhMain.cpp`: Implements `Dblqh` functionality (class methods).
- This directory also has the files listed here in a `redoLogReader` subdirectory containing the sources for the `ndbd_redo_log_reader` utility (see `ndbd_redo_log_reader` — Check and Print Content of Cluster Redo Log):
  - `records.cpp`
  - `records.hpp`
  - `redoLogFileReader.cpp`

This block also handles redo logging, and helps oversee the `DBACC`, `DBTUP`, `LGMAN`, `TSMAN`, `PGMAN`, and `BACKUP` blocks. It is implemented as the class `Dblqh`, defined in the file `storage/ndb/src/kernel/blocks/dblqh/Dblqh.hpp`.

4.8 The DBSPJ Block

This block implements multiple cursors in the NDB kernel, providing handling for joins pushed down from SQL nodes. It contains the following files, which can be found in the directory `storage/ndb/src/kernel/blocks/dbspj`:

- `Dbspj.hpp`: Defines the `Dbspj` class.
- `DbspjInit.cpp`: `Dbspj` initialization.
- `DbspjMain.cpp`: Handles conditions pushed down from API and signal passing between `DBSPJ` and the `DBLQH` and `DBTC` kernel blocks.
- `DbspjProxy.hpp`
- `DbspjProxy.cpp`

4.9 The DBTC Block

This is the transaction coordinator block, which handles distributed transactions and other data operations on a global level (as opposed to `DBLQH` which deals with such issues on individual data nodes). In the source code, it is located in the directory `storage/ndb/src/kernel/blocks/dbtc`, which contains these files:

- `Dbtc.hpp`: Defines the `Dbtc` class and associated constructs, including the following:
  - **Trigger and index data (TcDefinedTriggerData).** A record forming a list of active triggers for each table. These records are managed by a trigger pool, in which a trigger record is seized whenever a trigger is activated, and released when the trigger is deactivated.
  - **Fired trigger data (TcFiredTriggerData).** A record forming a list of fired triggers for a given transaction.
  - **Index data (TcIndexData).** This record forms lists of active indexes for each table. Such records are managed by an index pool, in which each index record is seized whenever an index is created, and released when the index is dropped.
  - **API connection record (ApiConnectRecord).** An API connect record contains the connection record to which the application connects. The application can send one operation at a time. It can

send a new operation immediately after sending the previous operation. This means that several operations can be active in a single transaction within the transaction coordinator, which is achieved by using the API connect record. Each active operation is handled by the TC connect record; as soon as the TC connect record has sent the request to the local query handler, it is ready to receive new operations. The `LQH` connect record takes care of waiting for an operation to complete; when an operation has completed on the `LQH` connect record, a new operation can be started on the current `LQH` connect record. `ApiConnectRecord` is always 256-byte aligned.

- **Transaction coordinator connection record (TcConnectRecord).** A `TcConnectRecord`) keeps all information required for carrying out a transaction; the transaction controller establishes connections to the different blocks needed to carry out the transaction. There can be multiple records for each active transaction. The TC connection record cooperates with the API connection record for communication with the API node, and with the `LQH` connection record for communication with any local query handlers involved in the transaction. `TcConnectRecord`) is permanently connected to a record in `DBDICT` and another in `DIH`, and contains a list of active `LQH` connection records and a list of started (but not currently active) `LQH` connection records. It also contains a list of all operations that are being executed with the current TC connection record. `TcConnectRecord` is always 128-byte aligned.

- **Cache record (CacheRecord).** This record is used between reception of a `TCKEYREQ` and sending of `LQHKEYREQ` (see Section 3.3, “Operations and Signals”) This is a separate record, so as to improve the cache hit rate and as well as to minimize memory storage requirements.

- **Host record (HostRecord).** This record contains the “alive” status of each node in the system, and is 128-byte aligned.

- **Table record (TableRecord).** This record contains the current schema versions of all tables in the system.

- **Scan record (ScanRecord).** Each scan allocates a `ScanRecord` to store information about the current scan.

- **Data buffer (DatabufRecord).** This is a buffer used for general data storage.

- **Attribute information record (AttrbufRecord).** This record can contain one (1) `ATTRINFO` signal, which contains a set of 32 attribute information words.

- **Global checkpoint information record (GcpRecord).** This record is used to store the globalcheckpoint number, as well as a counter, during the completion phase of the transaction. A `GcpRecord` is 32-byte aligned.

- **TC failure record (TC_FAIL_RECORD).** This is used when handling takeover of TC duties from a failed transaction coordinator.

- `DbtcInit.cpp`: Handles allocation and deallocation of `Dbtc` indexes and data (includes class desctructor).

- `DbtcMain.cpp`: Implements `Dbtc` methods.

Note

Any data node may act as the transaction coordinator.

The `DBTC` block is implemented as the `Dbtc` class.

The transaction coordinator is the kernel interface to which applications send their requests. It establishes connections to different blocks in the system to carry out the transaction and decides which node will handle each transaction, sending a confirmation signal on the result to the application so that the application can verify that the result received from the TUP block is correct.

This block also handles unique indexes, which must be co-ordinated across all data nodes simultaneously.

4.10 The DBTUP Block

This is the tuple manager, which manages the physical storage of cluster data. It consists of the following files found in the directory `storage/ndb/src/kernel/blocks/dbtup`:

- `AttributeOffset.hpp`: Defines the `AttributeOffset` class, which models the structure of an attribute, permitting up to 4096 attributes, all of which are nullable.
- `DbtupDiskAlloc.cpp`: Handles allocation and deallocation of extents for disk space.
- `DbtupIndex.cpp`: Implements methods for reading and writing tuples using ordered indexes.
- `DbtupScan.cpp`: Implements methods for tuple scans.
- `tuppage.cpp`: Handles allocating pages for writing tuples.
- `tuppage.hpp`: Defines structures for fixed and variable size data pages for tuples.
- `DbtupAbort.cpp`: Contains routines for terminating failed tuple operations.
- `DbtupExecQuery.cpp`: Handles execution of queries for tuples and reading from them.
- `DbtupMeta.cpp`: Handle table operations for the `Dbtup` class.
- `DbtupStoredProcDef.cpp`: Module for adding and dropping procedures.
- `DbtupBuffer.cpp`: Handles read/write buffers for tuple operations.
- `DbtupFixAlloc.cpp`: Allocates and frees fixed-size tuples from the set of pages attatched to a fragment. The fixed size is set per fragment; there can be only one such value per fragment.
- `DbtupPageMap.cpp`: Routines used by `Dbtup` to map logical page IDs to physical page IDs. The mapping needs the fragment ID and the logical page ID to provide the physical ID. This part of `Dbtup` is the exclusive user of a certain set of variables on the fragment record; it is also the exclusive user of the struct for page ranges (the `PageRange` struct defined in `Dbtup.hpp`).
- `DbtupTabDesMan.cpp`: This file contains the routines making up the table descriptor memory manager. Each table has a descriptor, which is a contiguous array of data words, and which is allocated from a global array using a “buddy” algorithm, with free lists existing for each 2^N words.
- `Notes.txt`: Contains some developers' implementation notes on tuples, tuple operations, and tuple versioning.
- `Undo_buffer.hpp`: Defines the `Undo_buffer` class, used for storage of operations that may need to be rolled back.
- `Undo_buffer.cpp`: Implements some necessary `Undo_buffer` methods.
- `DbtupCommit.cpp`: Contains routines used to commit operations on tuples to disk.
- `DbtupGen.cpp`: This file contains `Dbtup` initialization routines.

- `DbtupPagMan.cpp`: This file implements the page memory manager's “buddy” algorithm. `PagMan` is invoked when fragments lack sufficient internal page space to accommodate all the data they are requested to store. It is also invoked when fragments deallocate page space back to the free area.
- `DbtupTrigger.cpp`: The routines contained in this file perform handling of `NDB` internal triggers.
- `DbtupDebug.cpp`: Used for debugging purposes only.
- `Dbtup.hpp`: Contains the `Dbtup` class definition. Also defines a number of essential structures such as tuple scans, disk allocation units, fragment records, and so on.
- `DbtupRoutines.cpp`: Implements `Dbtup` routines for reading attributes.
- `DbtupVarAlloc.cpp`
- `test_varpage.cpp`: Simple test program for verifying variable-size page operations.

This block also monitors changes in tuples.

4.11 The DBTUX Block

This kernel block provides local management of ordered indexes. It consists of the following files found in the `storage/ndb/src/kernel/blocks/dbtux` directory:

- `DbtuxCmp.cpp`: Implements routines to search by key versus node prefix or entry. The comparison starts at a given attribute position, which is updated by the number of equal initial attributes found. The entry data may be partial, in which case `CmpUnknown` may be returned. The attributes are normalized and have a variable size, given in words.
- `DbtuxGen.cpp`: Implements initialization routines used in node starts and restarts.
- `DbtuxMaint.cpp`: Contains routines used to maintain indexes.
- `DbtuxNode.cpp`: Implements routines for node creation, allocation, and deletion operations. Also assigns lists of scans to nodes.
- `DbtuxSearch.cpp`: Provides routines for handling node scan request messages.
- `DbtuxTree.cpp`: Routines for performing node tree operations.
- `Times.txt`: Contains some (old) performance figures from tests runs on operations using ordered indexes. Of historical interest only.
- `DbtuxDebug.cpp`: Debugging code for dumping node states.
- `Dbtux.hpp`: Contains `Dbtux` class definition.
- `DbtuxMeta.cpp`: Routines for creating, setting, and dropping indexes. Also provides means of aborting these operations in the event of failure.
- `DbtuxScan.cpp`: Routines for performing index scans.
- `DbtuxStat.cpp`: Implements methods for obtaining node statistics.
- `tuxstatus.html`: 2004-01-30 status report on ordered index implementation. Of historical interest only.

4.12 The DBUTIL Block

This block provides internal interfaces to transaction and data operations, performing essential operations on signals passed between nodes. This block implements transactional services which can then be used by other blocks. It is also used in building online indexes, and is found in `storage/ndb/src/kernel/blocks/dbutil`, which includes these files:

- `DbUtil.cpp`: Implements `Dbutil` class methods
- `DbUtil.hpp`: Defines the `Dbutil` class, used to provide transactional services.
- `DbUtil.txt`: Implementation notes on utility protocols implemented by `DBUTIL`.

Among the duties performed by this block is the maintenance of sequences for backup IDs and other distributed identifiers.

4.13 The LGMAN Block

This block, the log group manager, is responsible for handling the undo logs for Disk Data tables. It is implemented in these files in the `storage/ndb/src/kernel/blocks` directory:

- `lgman.cpp`: Implements `Lgman` for adding, dropping, and working with log files and file groups.
- `lgman.hpp`: Contains the definition for the `Lgman` class, used to handle undo log files. Handles allocation of log buffer space.

4.14 The NDBCNTR Block

This is a cluster management block that handles block initialisation and configuration. During the data node startup process, it takes over from the `QMGR` block and continues the process. It also assists with graceful (planned) shutdowns of data nodes. This block is implemented in `storage/ndb/src/kernel/blocks/ndbcntr`, which contains these files:

- `Ndbcntr.hpp`: Defines the `Ndbcntr` class used to implement cluster management functions.
- `NdbcntrInit.cpp`: Initializers for `Ndbcntr` data and records.
- `NdbcntrMain.cpp`: Implements methods used for starts, restarts, and reading of configuration data.
- `NdbcntrSysTable.cpp`: `NDBCNTR` creates and initializes system tables on initial system start. The tables are defined in static structs in this file.

4.15 The NDBFS Block

This block provides the `NDB` file system abstraction layer, and is located in the directory `storage/ndb/src/kernel/blocks/ndbfs`, which contains the following files:

- `AsyncFile.hpp`: Defines the `AsyncFile` class, which represents an asynchronous file. All actions are executed concurrently with the other activities carried out by the process. Because all actions are performed in a separate thread, the result of an action is sent back through a memory channel. For the asyncronous notification of a finished request, each callincludes a request as a parameter. This class is used for writing or reading data to and from disk concurrently with other activities.
- `AsyncFile.cpp`: Defines the actions possible for an asynchronous file, and implements them.
- `Filename.hpp`: Defines the `Filename` class. Takes a 128-bit value (as a array of four longs) and makes a file name out of it. This file name encodes information about the file, such as whether it is a file

or a directory, and if the former, the type of file. Possible types include data file, fragment log, fragment list, table list, schema log, and system file, among others.

- `Filename.cpp`: Implements `set()` methods for the `Filename` class.
- `MemoryChannelTest/MemoryChannelTest.cpp`: Basic program for testing reads from and writes to a memory channel (that is, reading from and writing to a circular buffer).
- `OpenFiles.hpp`: Implements an `OpenFiles` class, which provides some convenience methods for determining whether or not a given file is already open.
- `VoidFs.cpp`: Used for diskless operation. Generates a “dummy” acknowledgment to write operations.
- `CircularIndex.hpp`: The `CircularIndex` class, defined in this file, serves as the building block for implementing circular buffers. It increments as a normal index until it reaches maximum size, then resets to zero.
- `CircularIndex.cpp`: Contains only a single `#define`, not actually used at this time.
- `MemoryChannel.hpp`: Defines the `MemoryChannel` and `MemoryChannelMultipleWriter` classes, which provide a pointer-based channel for communication between two threads. It does not copy any data into or out of the channel, so the item that is put in can not be used untill the other thread has given it back. There is no support for detecting the return of an item.
- `MemoryChannel.cpp`: “Dummy” file, not used at this time.
- `Ndbfs.hpp`: Because an `NDB` signal request can result in multiple requests to `AsyncFile`, one class (defined in this file) is responsible for keeping track of all outstanding requests, and when all are finished, reporting the outcome back to the sending block.
- `Ndbfs.cpp`: Implements initialization and signal-handling methods for the `Ndbfs` class.
- `Pool.hpp`: Creates and manages a pool of objects for use by `Ndbfs` and other classes in this block.
- `AsyncFileTest/AsyncFileTest.cpp`: Test program, used to test and benchmark functionality of `AsyncFile`.

4.16 The PGMAN Block

This block provides page and buffer management services for Disk Data tables. It includes these files:

- `diskpage.hpp`: Defines the `File_formats`, `Datafile`, and `Undofile` structures.
- `diskpage.cpp`: Initializes sero page headers; includes some output reoutines for reporting and debugging.
- `pgman.hpp`: Defines the `Pgman` class implementing a number of page and buffer services, including page entries and requests, page replacement, page lists, page cleanup, and other page processing.
- `pgman.cpp`: Implements `Pgman` methods for initialization and various page management tasks.
- `PgmanProxy.hpp`
- `PgmanProxy.cpp`

4.17 The QMGR Block

This is the logical cluster management block, and handles node membership in the cluster using a heartbeat mechanism. `QMGR` is responsible for polling the data nodes when a data node failure occurs and

determining that the node has actually failed and should be dropped from the cluster. This block contains the following files, found in `storage/ndb/src/kernel/blocks/qmgr`:

- `Qmgr.hpp`: Defines the Qmgr class and associated structures, including those used in detection of node failure and cluster partitioning.
- `QmgrInit.cpp`: Implements data and record initilization methods for `Qmgr`, as well as its destructor.
- `QmgrMain.cpp`: Contains routines for monitoring of heartbeats, detection and handling of “split-brain” problems, and management of some startup phases.
- `timer.hpp`: Defines the `Timer` class, used by `NDB` to keep strict timekeeping independent of the system clock.

This block also assists in the early phases of data node startup.

The `QMGR` block is implemented by the `Qmgr` class, whose definition is found in the file `storage/ndb/src/kernel/blocks/qmgr/Qmgr.hpp`.

4.18 The RESTORE Block

This block is implemented in the files `restore.hpp`, `restore.cpp`, `RestoreProxy.hpp`, and `RestoreProxy.cpp` in the `storage/ndb/src/kernel/blocks` directory. It handles restoration of the cluster from online backups. It is also used to restore local checkpoints as part of the process of starting a data node.

4.19 The SUMA Block

The cluster subscription manager, which handles event logging and reporting functions. It also figures prominently in NDB Cluster Replication. `SUMA` consists of the following files, found in the directory `storage/ndb/src/kernel/blocks/suma/`:

- `Suma.hpp`: Defines the `Suma` class and interfaces for managing subscriptions and performing necessary communications with other `SUMA` (and other) blocks.
- `SumaInit.cpp`: Performs initialization of `DICT`, `DIH`, and other interfaces
- `Suma.cpp`: Implements subscription-handling routines.
- `Suma.txt`: Contains a text-based diagram illustrating `SUMA` protocols.

4.20 The THRMAN Block

This is the thread management block, and executes in every NDB kernel thread. This block is also used measure thread CPU usage and to write this and other information into the `threadblocks` and `threadstat` tables in the `ndbinfo` information database.

The `THRMAN` block is implemented as the `Thrman` class, in the file `storage/ndb/src/kernel/blocks/thrman.hpp`. `thrman.cpp`, found in the same directory, defines a `measure_cpu_usage()` method of this class for measuring the CPU usage of a given thread. It also defines a `execDBINFO_SCANREQ()` method, which writes this and other information about the thread such as its thread ID and block number to the `threadblocks` and `threadstat` tables.

4.21 The TRPMAN Block

This is the signal transport management block of the NDB kernel, implemented in storage/ndb/src/kernel/blocks/trpman.hpp as the Trpman class, whose methods are defined in trpman.cpp, also in the blocks directory.

TRPMAN is also responsible for writing rows to the ndbinfo.transporters table.

4.22 The TSMAN Block

This is the tablespace manager block for Disk Data tables, implemented in the following files from storage/ndb/src/kernel/blocks:

- tsman.hpp: Defines the Tsman class, as well as structures representing data files and tablespaces.
- tsman.cpp: Implements Tsman methods.

4.23 The TRIX Block

This kernel block is responsible for the handling of internal triggers and unique indexes. TRIX, like DBUTIL, is a utility block containing many helper functions for building indexes and handling signals between nodes. It is implemented in the following files, all found in the directory storage/ndb/src/kernel/blocks/trix:

- Trix.hpp: Defines the Trix class, along with structures representing subscription data and records (for communicating with SUMA) and node data and ists (needed when communicating with remote TRIX blocks).
- Trix.cpp: Implements Trix class methods, including those necessary for taking appropriate action in the event of node failures.

Chapter 5 NDB Cluster Start Phases

Table of Contents

5.1 Initialization Phase (Phase -1) 91
5.2 Configuration Read Phase (STTOR Phase -1) 92
5.3 STTOR Phase 0 93
5.4 STTOR Phase 1 94
5.5 STTOR Phase 2 97
5.6 NDB_STTOR Phase 1 97
5.7 STTOR Phase 3 97
5.8 NDB_STTOR Phase 2 97
5.9 STTOR Phase 4 98
5.10 NDB_STTOR Phase 3 98
5.11 STTOR Phase 5 99
5.12 NDB_STTOR Phase 4 99
5.13 NDB_STTOR Phase 5 99
5.14 NDB_STTOR Phase 6 100
5.15 STTOR Phase 6 100
5.16 STTOR Phase 7 101
5.17 STTOR Phase 8 101
5.18 NDB_STTOR Phase 7 101
5.19 STTOR Phase 9 101
5.20 STTOR Phase 101 101
5.21 System Restart Handling in Phase 4 101
5.22 START_MEREQ Handling 102

The start of an NDB Cluster data node is processed in series of phases which is synchronised with other nodes that are starting up in parallel with this node as well as with nodes already started. The next several sections of this chapter describe each of these phases in detail.

5.1 Initialization Phase (Phase -1)

Before the data node actually starts, a number of other setup and initialization tasks must be done for the block objects, transporters, and watchdog checks, among others.

This initialization process begins in `storage/ndb/src/kernel/main.cpp` with a series of calls to `globalEmulatorData.theThreadConfig->doStart()`. When starting `ndbd` with the `-n` or `--nostart` option there is only one call to this method; otherwise, there are two, with the second call actually starting the data node. The first invocation of `doStart()` sends the `START_ORD` signal to the `CMVMI` block (see Section 4.2, “The CMVMI Block”); the second call to this method sends a `START_ORD` signal to `NDBCNTR` (see Section 4.14, “The NDBCNTR Block”).

When `START_ORD` is received by the `NDBCNTR` block, the signal is immediately transferred to `NDBCNTR`'s `MISSRA` sub-block, which handles the start process by sending a `READ_CONFIG_REQ` signals to all blocks in order as given in the array `readConfigOrder`:

1. `NDBFS`
2. `DBTUP`
3. `DBACC`

4. DBTC
5. DBLQH
6. DBTUX
7. DBDICT
8. DBDIH
9. NDBCNTR
10. QMGR
11. TRIX
12. BACKUP
13. DBUTIL
14. SUMA
15. TSMAN
16. LGMAN
17. PGMAN
18. RESTORE

NDBFS is permitted to run before any of the remaining blocks are contacted, in order to make sure that it can start the CMVMI block's threads.

5.2 Configuration Read Phase (STTOR Phase -1)

The READ_CONFIG_REQ signal provides all kernel blocks an opportunity to read the configuration data, which is stored in a global object accessible to all blocks. All memory allocation in the data nodes takes place during this phase.

Note

Connections between the kernel blocks and the NDB file system are also set up during Phase 0. This is necessary to enable the blocks to communicate easily which parts of a table structure are to be written to disk.

NDB performs memory allocations in two different ways. The first of these is by using the allocRecord() method (defined in storage/ndb/src/kernel/vm/SimulatedBlock.hpp). This is the traditional method whereby records are accessed using the ptrCheckGuard macros (defined in storage/ndb/src/kernel/vm/pc.hpp). The other method is to allocate memory using the setSize() method defined with the help of the template found in storage/ndb/src/kernel/vm/CArray.hpp.

These methods sometimes also initialize the memory, ensuring that both memory allocation and initialization are done with watchdog protection.

Many blocks also perform block-specific initialization, which often entails building linked lists or doubly-linked lists (and in some cases hash tables).

Many of the sizes used in allocation are calculated in the `Configuration::calcSizeAlt()` method, found in `storage/ndb/src/kernel/vm/Configuration.cpp`.

Some preparations for more intelligent pooling of memory resources have been made. `DataMemory` and disk records already belong to this global memory pool.

5.3 STTOR Phase 0

Most `NDB` kernel blocks begin their start phases at `STTOR` Phase 1, with the exception of `NDBFS` and `NDBCNTR`, which begin with Phase 0, as can be seen by inspecting the first value for each element in the `ALL_BLOCKS` array (defined in `src/kernel/blocks/ndbcntr/NdbcntrMain.cpp`). In addition, when the `STTOR` signal is sent to a block, the return signal `STTORRY` always contains a list of the start phases in which the block has an interest. Only in those start phases does the block actually receive a `STTOR` signal.

`STTOR` signals are sent out in the order in which the kernel blocks are listed in the `ALL_BLOCKS` array. While `NDBCNTR` goes through start phases 0 to 255, most of these are empty.

Both activities in Phase 0 have to do with initialization of the `NDB` file system. First, if necessary, `NDBFS` creates the file system directory for the data node. In the case of an initial start, `NDBCNTR` clears any existing files from the directory of the data node to ensure that the `DBDIH` block does not subsequently discover any system files (if `DBDIH` were to find any system files, it would not interpret the start correctly as an initial start). (See also Section 4.5, “The DBDIH Block”.)

Each time that `NDBCNTR` completes the sending of one start phase to all kernel blocks, it sends a `NODE_STATE_REP` signal to all blocks, which effectively updates the `NodeState` in all blocks.

Each time that `NDBCNTR` completes a nonempty start phase, it reports this to the management server; in most cases this is recorded in the cluster log.

Finally, after completing all start phases, `NDBCNTR` updates the node state in all blocks using a `NODE_STATE_REP` signal; it also sends an event report advising that all start phases are complete. In addition, all other cluster data nodes are notified that this node has completed all its start phases to ensure all nodes are aware of one another's state. Each data node sends a `NODE_START_REP` to all blocks; however, this is significant only for `DBDIH`, so that it knows when it can unlock the lock for schema changes on `DBDICT`.

Note

In the following table, and throughout this text, we sometimes refer to `STTOR` start phases simply as “start phases” or “Phase *N*” (where *N* is some number). `NDB_STTOR` start phases are always qualified as such, and so referred to as “`NDB_STTOR` start phases” or “`NDB_STTOR` phases”.

Kernel Block	Receptive Start Phases
`NDBFS`	0
`DBTC`	1
`DBDIH`	1
`DBLQH`	1, 4
`DBACC`	1
`DBTUP`	1
`DBDICT`	1, 3

Kernel Block	Receptive Start Phases
NDBCNTR	0, 1, 2, 3, 4, 5, 6, 8, 9
CMVMI	1 (prior to QMGR), 3, 8
QMGR	1, 7
TRIX	1
BACKUP	1, 3, 7
DBUTIL	1, 6
SUMA	1, 3, 5, 7, 100 (empty), 101
DBTUX	1,3,7
TSMAN	1, 3 (both ignored)
LGMAN	1, 2, 3, 4, 5, 6 (all ignored)
PGMAN	1, 3, 7 (Phase 7 currently empty)
RESTORE	1,3 (only in Phase 1 is any real work done)

Note

This table was current at the time this text was written, but is likely to change over time. The latest information can be found in the source code.

5.4 STTOR Phase 1

This is one of the phases in which most kernel blocks participate (see the table in Section 5.3, “STTOR Phase 0”). Otherwise, most blocks are involved primarily in the initialization of data—for example, this is all that DBTC does.

Many blocks initialize references to other blocks in Phase 1. DBLQH initializes block references to DBTUP, and DBACC initializes block references to DBTUP and DBLQH. DBTUP initializes references to the blocks DBLQH, TSMAN, and LGMAN.

NDBCNTR initializes some variables and sets up block references to DBTUP, DBLQH, DBACC, DBTC, DBDIH, and DBDICT; these are needed in the special start phase handling of these blocks using NDB_STTOR signals, where the bulk of the node startup process actually takes place.

If the cluster is configured to lock pages (that is, if the LockPagesInMainMemory configuration parameter has been set), CMVMI handles this locking.

The QMGR block calls the initData() method (defined in storage/ndb/src/kernel/blocks/qmgr/QmgrMain.cpp) whose output is handled by all other blocks in the READ_CONFIG_REQ phase (see Section 5.1, “Initialization Phase (Phase -1)”). Following these initializations, QMGR sends the DIH_RESTARTREQ signal to DBDIH, which determines whether a proper system file exists; if it does, an initial start is not being performed. After the reception of this signal comes the process of integrating the node among the other data nodes in the cluster, where data nodes enter the cluster one at a time. The first one to enter becomes the master; whenever the master dies the new master is always the node that has been running for the longest time from those remaining.

QMGR sets up timers to ensure that inclusion in the cluster does not take longer than what the cluster's configuration is set to permit (see Controlling Timeouts, Intervals, and Disk Paging for the relevant configuration parameters), after which communication to all other data nodes is established. At this point, a CM_REGREQ signal is sent to all data nodes. Only the president of the cluster responds to this signal;

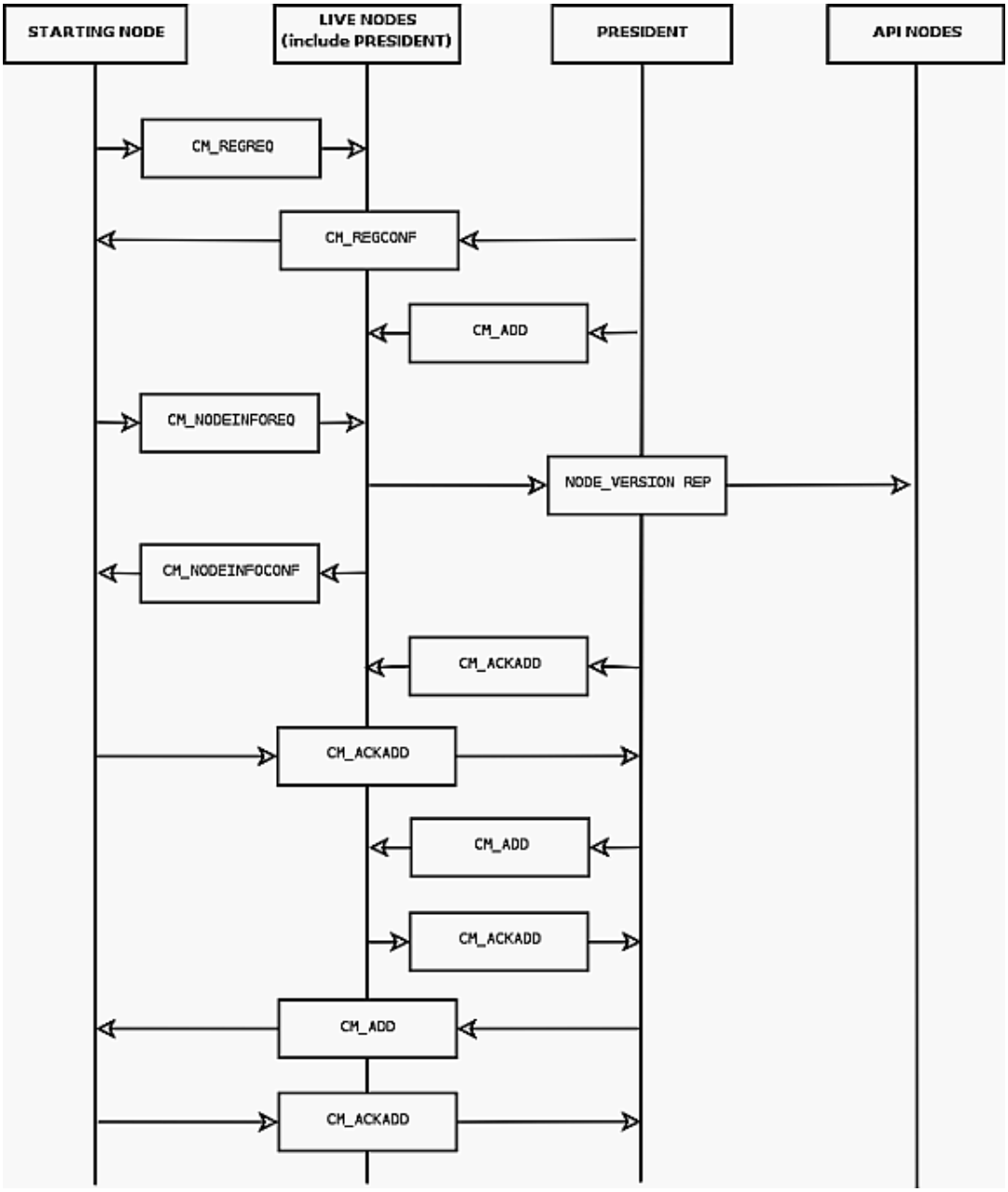
the president permits one node at a time to enter the cluster. If no node responds within 3 seconds then the president becomes the master. If several nodes start up simultaneously, then the node with the lowest node ID becomes president. The president sends `CM_REGCONF` in response to this signal, but also sends a `CM_ADD` signal to all nodes that are currently alive.

Next, the starting node sends a `CM_NODEINFOREQ` signal to all current “live” data nodes. When these nodes receive that signal they send a `NODE_VERSION_REP` signal to all API nodes that have connected to them. Each data node also sends a `CM_ACKADD` to the president to inform the president that it has heard the `CM_NODEINFOREQ` signal from the new node. Finally, each of the current data nodes sends the `CM_NODEINFOCONF` signal in response to the starting node. When the starting node has received all these signals, it also sends the `CM_ACKADD` signal to the president.

When the president has received all of the expected `CM_ACKADD` signals, it knows that all data nodes (including the newest one to start) have replied to the `CM_NODEINFOREQ` signal. When the president receives the final `CM_ACKADD`, it sends a `CM_ADD` signal to all current data nodes (that is, except for the node that just started). Upon receiving this signal, the existing data nodes enable communication with the new node; they begin sending heartbeats to it and including in the list of neighbors used by the heartbeat protocol.

The `start` struct is reset, so that it can handle new starting nodes, and then each data node sends a `CM_ACKADD` to the president, which then sends a `CM_ADD` to the starting node after all such `CM_ACKADD` signals have been received. The new node then opens all of its communication channels to the data nodes that were already connected to the cluster; it also sets up its own heartbeat structures and starts sending heartbeats. It also sends a `CM_ACKADD` message in response to the president.

The signalling between the starting data node, the already “live” data nodes, the president, and any API nodes attached to the cluster during this phase is shown in the following diagram:



As a final step, QMGR also starts the timer handling for which it is responsible. This means that it generates a signal to blocks that have requested it. This signal is sent 100 times per second even if any one instance of the signal is delayed..

The BACKUP kernel block also begins sending a signal periodically. This is to ensure that excessive amounts of data are not written to disk, and that data writes are kept within the limits of what has been specified in the cluster configuration file during and after restarts. The DBUTIL block initializes the transaction identity, and DBTUX creates a reference to the DBTUP block, while PGMAN initializes pointers to the LGMAN and DBTUP blocks. The RESTORE kernel block creates references to the DBLQH and DBTUP blocks to enable quick access to those blocks when needed.

5.5 STTOR Phase 2

The only kernel block that participates in this phase to any real effect is NDBCNTR.

In this phase NDBCNTR obtains the current state of each configured cluster data node. Messages are sent to NDBCNTR from QMGR reporting the changes in status of any the nodes. NDBCNTR also sets timers corresponding to the StartPartialTimeout, StartPartitionTimeout, and StartFailureTimeout configuration parameters.

The next step is for a CNTR_START_REQ signal to be sent to the proposed master node. Normally the president is also chosen as master. However, during a system restart where the starting node has a newer global checkpoint than that which has survived on the president, then this node will take over as master node, even though it is not recognized as the president by QMGR. If the starting node is chosen as the new master, then the other nodes are informed of this using a CNTR_START_REF signal.

The master withholds the CNTR_START_REQ signal until it is ready to start a new node, or to start the cluster for an initial restart or system restart.

When the starting node receives CNTR_START_CONF, it starts the NDB_STTOR phases, in the following order:

1. DBLQH
2. DBDICT
3. DBTUP
4. DBACC
5. DBTC
6. DBDIH

5.6 NDB_STTOR Phase 1

DBDICT, if necessary, initializes the schema file. DBDIH, DBTC, DBTUP, and DBLQH initialize variables. DBLQH also initializes the sending of statistics on database operations.

5.7 STTOR Phase 3

DBDICT initializes a variable that keeps track of the type of restart being performed.

NDBCNTR executes the second of the NDB_STTOR start phases, with no other NDBCNTR activity taking place during this STTOR phase.

5.8 NDB_STTOR Phase 2

The DBLQH block enables its exchange of internal records with DBTUP and DBACC, while DBTC permits its internal records to be exchanged with DBDIH. The DBDIH kernel block creates the mutexes used by the NDB kernel and reads nodes using the READ_NODESREQ signal. With the data from the response to this signal, DBDIH can create node lists, node groups, and so forth. For node restarts and initial node restarts, DBDIH also asks the master for permission to perform the restart. The master will ask all “live” nodes if they are prepared to permit the new node to join the cluster. If an initial node restart is to be performed, then all LCPs are invalidated as part of this phase.

LCPs from nodes that are not part of the cluster at the time of the initial node restart are not invalidated. The reason for this is that there is never any chance for a node to become master of a system restart using any of the LCPs that have been invalidated, since this node must complete a node restart—including a local checkpoint—before it can join the cluster and possibly become a master node.

The CMVMI kernel block activates the sending of packed signals, which occurs only as part of database operations. Packing must be enabled prior to beginning any such operations during the execution of the redo log or node recovery phases.

The DBTUX block sets the type of start currently taking place, while the BACKUP block sets the type of restart to be performed, if any (in each case, the block actually sets a variable whose value reflects the type of start or restart). The SUMA block remains inactive during this phase.

The PGMAN kernel block starts the generation of two repeated signals, the first handling cleanup. This signal is sent every 200 milliseconds. The other signal handles statistics, and is sent once per second.

5.9 STTOR Phase 4

Only the DBLQH and NDBCNTR kernel blocks are directly involved in this phase. DBLQH allocates a record in the BACKUP block, used in the execution of local checkpoints using the DEFINE_BACKUP_REQ signal. NDBCNTR causes NDB_STTOR to execute NDB_STTOR phase 3; there is otherwise no other NDBCNTR activity during this STTOR phase.

5.10 NDB_STTOR Phase 3

The DBLQH block initiates checking of the log files here. Then it obtains the states of the data nodes using the READ_NODESREQ signal. Unless an initial start or an initial node restart is being performed, the checking of log files is handled in parallel with a number of other start phases. For initial starts, the log files must be initialized; this can be a lengthy process and should have some progress status attached to it.

Note

From this point, there are two parallel paths, one continuing to restart and another reading and determining the state of the redo log files.

The DBDICT block requests information about the cluster data nodes using the READ_NODESREQ signal. DBACC resets the system restart flag if this is not a system restart; this is used only to verify that no requests are received from DBTUX during system restart. DBTC requests information about all nodes by means of the READ_NODESREQ signal.

DBDIH sets an internal master state and makes other preparations exclusive to initial starts. In the case of an initial start, the nonmaster nodes perform some initial tasks, the master node doing once all nonmaster nodes have reported that their tasks are completed. (This delay is actually unnecessary since there is no reason to wait while initializing the master node.)

For node restarts and initial node restarts no more work is done in this phase. For initial starts the work is done when all nodes have created the initial restart information and initialized the system file.

For system restarts this is where most of the work is performed, initiated by sending the NDB_STARTREQ signal from NDBCNTR to DBDIH in the master. This signal is sent when all nodes in the system restart have reached this point in the restart. This we can mark as our first synchronization point for system restarts, designated WAITPOINT_4_1.

For a description of the system restart version of Phase 4, see Section 5.21, “System Restart Handling in Phase 4”.

After completing execution of the NDB_STARTREQ signal, the master sends a CNTR_WAITREP signal with WAITPOINT_4_2 to all nodes. This ends NDB_STTOR phase 3 as well as (STTOR) Phase 4.

5.11 STTOR Phase 5

All that takes place in Phase 5 is the delivery by NDBCNTR of NDB_STTOR phase 4; the only block that acts on this signal is DBDIH that controls most of the part of a data node start that is database-related.

5.12 NDB_STTOR Phase 4

Some initialization of local checkpoint variables takes place in this phase, and for initial restarts, this is all that happens in this phase.

For system restarts, all required takeovers are also performed. Currently, this means that all nodes whose states could not be recovered using the redo log are restarted by copying to them all the necessary data from the “live” data nodes.

For node restarts and initial node restarts, the master node performs a number of services, requested to do so by sending the START_MEREQ signal to it. This phase is complete when the master responds with a START_MECONF message, and is described in Section 5.22, “START_MEREQ Handling”.

After ensuring that the tasks assigned to DBDIH tasks in the NDB_STTOR phase 4 are complete, NDBCNTR performs some work on its own. For initial starts, it creates the system table that keeps track of unique identifiers such as those used for AUTO_INCREMENT. Following the WAITPOINT_4_1 synchronization point, all system restarts proceed immediately to NDB_STTOR phase 5, which is handled by the DBDIH block. See Section 5.13, “NDB_STTOR Phase 5”, for more information.

5.13 NDB_STTOR Phase 5

For initial starts and system restarts this phase means executing a local checkpoint. This is handled by the master so that the other nodes will return immediately from this phase. Node restarts and initial node restarts perform the copying of the records from the primary replica to the starting replicas in this phase. Local checkpoints are enabled before the copying process is begun.

Copying the data to a starting node is part of the node takeover protocol. As part of this protocol, the node status of the starting node is updated; this is communicated using the global checkpoint protocol. Waiting for these events to take place ensures that the new node status is communicated to all nodes and their system files.

After the node's status has been communicated, all nodes are signaled that we are about to start the takeover protocol for this node. Part of this protocol consists of Steps 3 - 9 during the system restart phase as described below. This means that restoration of all the fragments, preparation for execution of the redo log, execution of the redo log, and finally reporting back to DBDIH when the execution of the redo log is completed, are all part of this process.

After preparations are complete, copy phase for each fragment in the node must be performed. The process of copying a fragment involves the following steps:

1. The DBLQH kernel block in the starting node is informed that the copy process is about to begin by sending it a PREPARE_COPY_FRAGREQ signal.

2. When DBLQH acknowledges this request a CREATE_FRAGREQ signal is sent to all nodes notify them of the preparation being made to copy data to this replica for this table fragment.

3. After all nodes have acknowledged this, a COPY_FRAGREQ signal is sent to the node from which the data is to be copied to the new node. This is always the primary replica of the fragment. The node indicated copies all the data over to the starting node in response to this message.
4. After copying has been completed, and a COPY_FRAGCONF message is sent, all nodes are notified of the completion through an UPDATE_TOREQ signal.
5. After all nodes have updated to reflect the new state of the fragment, the DBLQH kernel block of the starting node is informed of the fact that the copy has been completed, and that the replica is now up-to-date and any failures should now be treated as real failures.
6. The new replica is transformed into a primary replica if this is the role it had when the table was created.
7. After completing this change another round of CREATE_FRAGREQ messages is sent to all nodes informing them that the takeover of the fragment is now committed.
8. After this, process is repeated with the next fragment if another one exists.
9. When there are no more fragments for takeover by the node, all nodes are informed of this by sending an UPDATE_TOREQ signal sent to all of them.
10. Wait for the next complete local checkpoint to occur, running from start to finish.
11. The node states are updated, using a complete global checkpoint. As with the local checkpoint in the previous step, the global checkpoint must be permitted to start and then to finish.
12. When the global checkpoint has completed, it will communicate the successful local checkpoint of this node restart by sending an END_TOREQ signal to all nodes.
13. A START_COPYCONF is sent back to the starting node informing it that the node restart has been completed.
14. Receiving the START_COPYCONF signal ends NDB_STTOR phase 5. This provides another synchronization point for system restarts, designated as WAITPOINT_5_2.

Note

The copy process in this phase can in theory be performed in parallel by several nodes. However, all messages from the master to all nodes are currently sent to single node at a time, but can be made completely parallel. This is likely to be done in the not too distant future.

In an initial and an initial node restart, the SUMA block requests the subscriptions from the SUMA master node. NDBCNTR executes NDB_STTOR phase 6. No other NDBCNTR activity takes place.

5.14 NDB_STTOR Phase 6

In this NDB_STTOR phase, both DBLQH and DBDICT clear their internal representing the current restart type. The DBACC block resets the system restart flag; DBACC and DBTUP start a periodic signal for checking memory usage once per second. DBTC sets an internal variable indicating that the system restart has been completed.

5.15 STTOR Phase 6

The NDBCNTR block defines the cluster's node groups, and the DBUTIL block initializes a number of data structures to facilitate the sending keyed operations can be to the system tables. DBUTIL also sets up a single connection to the DBTC kernel block.

5.16 STTOR Phase 7

In QMGR the president starts an arbitrator (unless this feature has been disabled by setting the value of the ArbitrationRank configuration parameter to 0 for all nodes—see Defining an NDB Cluster Management Server, and Defining SQL and Other API Nodes in an NDB Cluster, for more information; note that this currently can be done only when using NDB Cluster Carrier Grade Edition). In addition, checking of API nodes through heartbeats is activated.

Also during this phase, the BACKUP block sets the disk write speed to the value used following the completion of the restart. The master node during initial start also inserts the record keeping track of which backup ID is to be used next. The SUMA and DBTUX blocks set variables indicating start phase 7 has been completed, and that requests to DBTUX that occurs when running the redo log should no longer be ignored.

5.17 STTOR Phase 8

NDB_STTOR executes NDB_STTOR phase 7; no other NDBCNTR activity takes place.

5.18 NDB_STTOR Phase 7

If this is a system restart, the master node initiates a rebuild of all indexes from DBDICT during this phase.

The CMVMI kernel block opens communication channels to the API nodes (including MySQL servers acting as SQL nodes). Indicate in globalData that the node is started.

5.19 STTOR Phase 9

NDBCNTR resets some start variables.

5.20 STTOR Phase 101

This is the SUMA handover phase, during which a GCP is negotiated and used as a point of reference for changing the source of event and replication subscriptions from existing nodes only to include a newly started node.

5.21 System Restart Handling in Phase 4

This consists of the following steps:

1. The master sets the latest GCI as the restart GCI, and then synchronizes its system file to all other nodes involved in the system restart.
2. The next step is to synchronize the schema of all the nodes in the system restart. This is performed in 15 passes. The problem we are trying to solve here occurs when a schema object has been created while the node was up but was dropped while the node was down, and possibly a new object was even created with the same schema ID while that node was unavailable. In order to handle this situation, it is necessary first to re-create all objects that are supposed to exist from the viewpoint of the starting node. After this, any objects that were dropped by other nodes in the cluster while this node was “dead” are dropped; this also applies to any tables that were dropped during the outage. Finally, any tables that have been created by other nodes while the starting node was unavailable are re-created on the starting node. All these operations are local to the starting node. As part of this process, is it also necessary to ensure that all tables that need to be re-created have been created locally and that the proper data structures have been set up for them in all kernel blocks.

   After performing the procedure described previously for the master node the new schema file is sent to all other participants in the system restart, and they perform the same synchronization.

3. All fragments involved in the restart must have proper parameters as derived from DBDIH. This causes a number of START_FRAGREQ signals to be sent from DBDIH to DBLQH. This also starts the restoration of the fragments, which are restored one by one and one record at a time in the course of reading the restore data from disk and applying in parallel the restore data read from disk into main memory. This restores only the main memory parts of the tables.

4. Once all fragments have been restored, a START_RECREQ message is sent to all nodes in the starting cluster, and then all undo logs for any Disk Data parts of the tables are applied.

5. After applying the undo logs in LGMAN, it is necessary to perform some restore work in TSMAN that requires scanning the extent headers of the tablespaces.

6. Next, it is necessary to prepare for execution of the redo log, which log can be performed in up to four phases. For each fragment, execution of redo logs from several different nodes may be required. This is handled by executing the redo logs in different phases for a specific fragment, as decided in DBDIH when sending the START_FRAGREQ signal. An EXEC_FRAGREQ signal is sent for each phase and fragment that requires execution in this phase. After these signals are sent, an EXEC_SRREQ signal is sent to all nodes to tell them that they can start executing the redo log.

   Note

   Before starting execution of the first redo log, it is necessary to make sure that the setup which was started earlier (in Phase 4) by DBLQH has finished, or to wait until it does before continuing.

7. Prior to executing the redo log, it is necessary to calculate where to start reading and where the end of the REDO log should have been reached. The end of the REDO log should be found when the last GCI to restore has been reached.

8. After completing the execution of the redo logs, all redo log pages that have been written beyond the last GCI to be restore are invalidated. Given the cyclic nature of the redo logs, this could carry the invalidation into new redo log files past the last one executed.

9. After the completion of the previous step, DBLQH report this back to DBDIH using a START_RECCONF message.

10. When the master has received this message back from all starting nodes, it sends a NDB_STARTCONF signal back to NDBCNTR.

11. The NDB_STARTCONF message signals the end of STTOR phase 4 to NDBCNTR, which is the only block involved to any significant degree in this phase.

5.22 START_MEREQ Handling

The first step in handling START_MEREQ is to ensure that no local checkpoint is currently taking place; otherwise, it is necessary to wait until it is completed. The next step is to copy all distribution information from the master DBDIH to the starting DBDIH. After this, all metadata is synchronized in DBDICT (see Section 5.21, “System Restart Handling in Phase 4”).

After blocking local checkpoints, and then synchronizing distribution information and metadata information, global checkpoints are blocked.

The next step is to integrate the starting node in the global checkpoint protocol, local checkpoint protocol, and all other distributed protocols. As part of this the node status is also updated.

After completing this step the global checkpoint protocol is permitted to start again, the START_MECONF signal is sent to indicate to the starting node that the next phase may proceed.

Chapter 6 NDB Schema Object Versions

NDB supports online schema changes. A schema object such as a Table or Index has a 4-byte *schema object version identifier*, which can be observed in the output of the ndb_desc utility (see ndb_desc — Describe NDB Tables), as shown here (emphasized text):

```
shell> ndb_desc -c 127.0.0.1 -d test t1
-- t1 --
Version: 33554434
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 3
Number of primary keys: 1
Length of frm data: 269
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
HashMap: DEFAULT-HASHMAP-240-4
-- Attributes --
c1 Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
c2 Int NULL AT=FIXED ST=MEMORY
c4 Varchar(50;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
-- Indexes --
PRIMARY KEY(c1) - UniqueHashIndex
PRIMARY(c1) - OrderedIndex

NDBT_ProgramExit: 0 - OK
```

The schema object version identifier (or simply “schema version”) is made up of a major version and a minor version; the major version occupies the (single) least sigificant byte of the schema version, and the minor version the remaining (3 most significant) bytes. You can see these two components more easily when viewing the schema version in hexadecimal notation. In the example output just shown, the schema version is shown as 33554434, which in hexadecimal (filling in leading zeroes as necessary) is 0x02000002; this is equivalent to major version 2, minor version 2. Adding an index to table t1 causes the schema version as reported by ndb_desc to advance to 50331650, or 0x03000002 hexadecimal, which is equivalent to major version 2 (3 least significant bytes 00 00 02), minor version 3 (most significant byte 03). Minor schema versions start with 0 for a newly created table.

In addition, each NDB API database object class has its own getObjectVersion() method that, like Object::getObjectVersion(), returns the object's schema object version. This includes instances, not only of Object, but of Table, Index, Column, LogfileGroup, Tablespace, Datafile, and Undofile, as well as Event. (However, NdbBlob::getVersion() has a purpose and function that is completely unrelated to that of the methods just listed.)

Schema changes which are considered backward compatible—such as adding a DEFAULT or NULL column at the end of a table—cause the table object's minor version to be incremented. Schema changes which are not considered backward compatible—such as removing a column from a table—cause the major version to be incremented.

Note

While the implementation of an operation causing a schema major version change may actually involve 2 copies of the affected table (dropping and recreating the table), the final outcome can be observed as an increase in the table's major version.

Queries and DML operations which arrive from NDB clients also have an associated schema version, which is checked at the start of processing in the data nodes. If the schema version of the request differs from the affected database object's latest schema version only in its minor version component, the operation is considered compatible and is allowed to proceed. If the schema version differs in the major schema version then it will be rejected.

This mechanism allows the schema to be changed in the data nodes in various ways, without requiring a synchronized schema change in clients. Clients need not move on to the new schema version until they are ready to do so. Queries and DML operations can thus continue uninterrupted.

The NDB API and schema object versions. An NDB API application normally uses an `NdbDictionary` object associated with an `Ndb` object to retrieve schema objects. Schema objects are retrieved on demand from the data nodes; signalling is used to obtain the table or index definition; then, a local memory object is constructed which the application can use. NDB internally caches schema objects, so that each successive request for the same table or index by name does not require signalling.

Global schema cache. To avoid the need to signal to the data nodes for every schema object lookup, a schema cache is used for each `Ndb_cluster_connection`. This is referred to as the *global schema cache*. It is global in terms of spanning multiple Ndb objects. Instantiated table and index objects are automatically put into this cache to save on future signalling and instantiation costs. The cache maintains a reference count for each object; this count is used to determine when a given schema object can be deleted. Schema objects can have their reference counts modified by explicit API method calls or local schema cache operations.

Local schema cache. In addition to the per-connection global schema cache, each `Ndb` object's `NdbDictionary` object has a *local schema cache*. This cache contains pointers to objects held in the global schema cache. Each local schema cache holding a reference to a schema object in the global schema cache increments the global schema cache reference count by 1. Having a schema cache that is local to each `Ndb` object allows schema objects to be looked up without imposing any locks. The local schema cache is normally emptied (reducing global cache reference counts in the process) when its associated `Ndb` object is deleted.

Operation without schema changes. Normal operation proceeds as follows in the cases listed below:

A. **A table is requested by some client (Ndb object) for the first time.** The local cache is checked; the attempt results in a miss. The global cache is then also checked (using a lock), and the result is another miss.

   Since there were no cache hits, the data node is sent a signal; the node's response is used to instantiate the table object. A pointer to the instantiated data object is added to the global cache; another such pointer is added to the local cache, and the reference count is set to 1. A pointer to the table is returned to the client.

B. **A second client (a different Ndb object) requests access to the same table, also by name.** A check of the local cache results in a miss, but a check of the global cache yields a hit.

   As a result, an object pointer is added to the local cache, the global reference count is incremented—so that its value is now 2—and an object pointer is returned to the client. No new pointer is added to the global cache.

C. **For a second time, the second client requests access to same table by name.** The local cache is checked, producing a hit. An object pointer is immediately returned to the client. No pointers are added to the local or global caches, and the object's reference count is not incremented (and so the reference count remains constant at 2).

D. **Second client deletes Ndb object.** Objects in this client's local schema cache have their reference counts decremented in global cache.

   This sets the global cache reference count to 1. Since it is not yet 0, no action is yet taken to remove the parent `Ndb` object.

Schema changes. Assuming that an object's schema never changes, the schema version first retrieved is used for the lifetime of the application process, and the in-memory object is deleted only when all local cache references (that is, all references to `Ndb` objects) have been deleted. This is unlikely to occur other than during a shutdown or cluster connection reset.

If an object's schema changes in a backward-compatible way while an application is running, this has the following affects:

- The minor version at the data nodes is incremented. (Ongoing DML operations using the old schema version still succeed.)
- NDB API clients subsequently retrieving the latest version of the schema object then fetch the new schema version.
- NDB API clients with cached older versions do not use the new schema version unless and until their local and global caches are invalidated.
- NDB API clients subscribing to events can observe a `TE_ALTER` event for the table in question, and can use this to trigger schema object cache invalidations.
- Each local cache entry can be removed by calling `removeCachedTable()` or `removeCachedIndex()`. This removes the entry from the local cache, and decrements the reference count in the global cache. When (and if) the global cache reference count reaches zero, the old cached object can be deleted.
- Alternatively, local cache entries can be removed, and the global cache entry invalidated, by calling `invalidateTable()` or `invalidateIndex()`. Subsequent calls to `getTable()` or `getIndex()` for this and other clients return the new schema object version by signalling the data nodes and instantiating a new object.
- New `Ndb` objects fill their local table caches on demand from the global table cache as normal. This means that, once an old schema object has been invalidated in the global cache, such objects retrieve the latest table objects known at the time that the table objects are first cached.

When an incompatible schema change is made (that is, a schema major version change), NDB API requests using the old version fail as soon as the new version is committed. This can also be used as a trigger to retrieve a new schema object version.

The rules governing the handling of schema version changes are summarized in the following list:

- An online schema change (minor version change) does not affect existing clients (`Ndb` objects); clients can continue to use the old schema object version
- If and only if a client voluntarily removes cached objects by making API calls can it then observe the new schema object version.

- As `Ndb` objects remove cached objects and are deleted, the reference count on the old schema object version decreases.
- When this reference count reaches 0, the object can be deleted.

Implications of the schema object lifecycle. The lifespan of a schema object (such as a `Table` or `Index`) is limited by the lifetime of the `Ndb` object from which it is obtained. When the parent `Ndb` object of a schema object is deleted, the reference count which keeps the `Ndb` object alive is decremented. If this `Ndb` object holds the last reamining reference to a given schema object version, the deletion of the `Ndb` object can also result in the deletion of the schema object. For this reason, no other threads can be using the object at this time.

Care must be exercised when pointers to schema objects are held in the application and used between multiple `Ndb` objects. A schema object should not be used beyond the lifespan of the `Ndb` object which created it.

Applications can respond, asynchronously and independently of each other, to backward-compatible schema changes, moving to the new schema only when necessary. Different threads can operate on different schema object versions concurrently.

It is thus very important to ensure that schema objects do not outlive the `Ndb` objects used to create them. To help prevent this from happening, you can take any of the following actions to invalidate old schema objects:

- To trigger invalidation when and as needed, use NDB API `TE_ALTER` events (see Event::TableEvent).
- Use an external trigger to initiate invalidation.
- Perform a periodic invalidation explicitly.

Invalidating the caches in any of these ways allows applications to obtain new versions of schema objects as required.

It is also worth noting that not all NDB API `Table` getter methods return pointers; many of them (in addition to `Table::getName()`) return table names. Such methods include `Index::getTable()`, `NdbOperation::getTableName()`, `Event::getTableName()`, and `NdbDictionary::getRecordTableName()`.

Chapter 7 NDB Cluster API Errors

Table of Contents

7.1 Data Node Error Messages 107
7.1.1 ndbd Error Codes 107
7.1.2 ndbd Error Classifications 111
7.2 NDB Transporter Errors 112

This section provides a listing of exit codes and messages returned by a failed data node (ndbd or ndbmtd) process, as well as NDB transporter error log messages.

For information about error handling and error codes for the NDB API, see NDB API Errors and Error Handling. For information about error handling and error codes for the MGM API, see MGM API Errors, as well as The ndb_mgm_error Type.

7.1 Data Node Error Messages

This section contains exit codes and error messages given when a data node process stops prematurely.

7.1.1 ndbd Error Codes

This section lists all the error messages that can be returned when a data node process halts due to an error, arranged in most cases according to the affected NDB kernel block.

For more information about kernel blocks, see Chapter 4, *NDB Kernel Blocks*

The meanings of the values given in the **Classification** column of each of the following tables is given in Section 7.1.2, “ndbd Error Classifications”.

7.1.1.1 General Errors

This section contains ndbd error codes that are either generic in nature or otherwise not associated with a specific NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_GENERIC	Error	Generic error
NDBD_EXIT_PROGERR	Error	Assertion
NDBD_EXIT_NODE_NOT_IN_CONFIG	Error	Node ID in the configuration has the wrong type (that is, it is not a data node)
NDBD_EXIT_SYSTEM_ERROR	Error	System error, node killed during node restart by other node
NDBD_EXIT_INDEX_NOTINRANGE	Error	Array index out of range
NDBD_EXIT_ARBIT_SHUTDOWN	Error	Node lost connection to other nodes and can not form a unpartitioned cluster, please investigate if there are error(s) on other node(s)
NDBD_EXIT_PARTITIONED_SHUTDOWN	Error	Partitioned cluster detected. Please check if cluster is already running
NDBD_EXIT_NODE_DECLARED_DEAD	Error	Node declared dead. See error log for details

Error Code	Error Classification	Error Text
NDBD_EXIT_POINTER_NOTINRANGE	XIE	Pointer too large
NDBD_EXIT_SR_OTHERNODEFAILED	XIE	Another node failed during system restart, please investigate error(s) on other node(s)
NDBD_EXIT_NODE_NOT_DEAD	XIE	Internal node state conflict, most probably resolved by restarting node again
NDBD_EXIT_SR_REDOLOG	XIE	Error while reading the REDO log
NDBD_EXIT_SR_SCHEMAFILE	XIE	Error while reading the schema file
2311	XIE	Conflict when selecting restart type
NDBD_EXIT_NO_MORE_UNDOLOG	XIE	No more free UNDO log, increase UndoIndexBuffer
NDBD_EXIT_SR_UNDOLOG	XIE	Error while reading the data pages and UNDO log
NDBD_EXIT_SINGLE_USER_MODE	XIE	Data node is not allowed to get added to the cluster while it is in single user mode
NDBD_EXIT_MEMALLOC	XIE	Memory allocation failure, please decrease some configuration parameters
NDBD_EXIT_BLOCK_JBUFCONGESTION	XIE	Job buffer congestion
NDBD_EXIT_TIME_QUEUE_SHORT	XIE	Error in short time queue
NDBD_EXIT_TIME_QUEUE_LONG	XIE	Error in long time queue
NDBD_EXIT_TIME_QUEUE_DELAY	XIE	Error in time queue, too long delay
NDBD_EXIT_TIME_QUEUE_INDEX	XIE	Time queue index out of range
NDBD_EXIT_BLOCK_BNR_ZERO	XIE	Send signal error
NDBD_EXIT_WRONG_PRIO_LEVEL	XIE	Wrong priority level when sending signal
NDBD_EXIT_NDBREQUIRE	XIE	Internal program error (failed ndbrequire)
NDBD_EXIT_NDBASSERT	XIE	Internal program error (failed ndbassert)
NDBD_EXIT_ERROR_INSERT	XIE	Error insert executed
NDBD_EXIT_INVALID_CONFIG	XIE	Invalid configuration received from Management Server
NDBD_EXIT_RESOURCE_ALLOC_ERROR	XIE	Resource allocation error, please review the configuration
NDBD_EXIT_NO_MORE_REDOLOG	XIE	Fatal error due to end of REDO log. Increase NoOfFragmentLogFiles or FragmentLogFileSize
NDBD_EXIT_OS_SIGNAL_RECEIVED	XIE	Error OS signal received
NDBD_EXIT_SR_RESTARTCONFLICT	XIE	Partial system restart causing conflicting file systems

7.1.1.2 VM Errors

This section contains ndbd error codes that are associated with problems in the VM (virtal machine) NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_OUT_OF_LONG_SIGNAL_MEMORY	XIE	Signal memory, out of long signal memory, please increase LongMessageBuffer

Error Code	Error Classification	Error Text
NDBD_EXIT_WATCHDOG_TERMINATE	ExitCode	WatchDog terminate, internal error or massive overload on the machine running this node
NDBD_EXIT_SIGNAL_LOST_SEND_BUFFER_FULL	Resource	Send buffer full, out of send buffer memory, please increase SendBufferMemory or lower the load
NDBD_EXIT_SIGNAL_LOST	Unknown	Signal lost (unknown reason)
NDBD_EXIT_ILLEGAL_SIGNAL	InternalError	Illegal signal (version mismatch a possibility)
NDBD_EXIT_CONNECTION_SETUP_FAILED	ExitCode	Connection setup failed

7.1.1.3 NDBCNTR Errors

This section contains ndbd error codes that are associated with problems in the NDBCNTR (initialization and configuration) NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_RESTART_TIMEOUT	StartupError	Total restart time too long, consider increasing StartFailureTimeout or investigate error(s) on other node(s)
NDBD_EXIT_RESTART_DURING_SHUTDOWN	StartupError	Node started while node shutdown in progress. Please wait until shutdown complete before starting node

7.1.1.4 DIH Errors

This section contains ndbd error codes that are associated with problems in the DIH (distribution handler) NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_MAX_CRASHED_REPLICAS	InternalError	Too many crashed replicas (8 consecutive node restart failures)
NDBD_EXIT_MASTER_FAILURE_DURING_NR	InternalError	Master failure during node restart
NDBD_EXIT_LOST_NODE_GROUP	InternalError	All nodes in a node group are unavailable
NDBD_EXIT_NO_RESTORABLE_REPLICA	InternalError	Unable to find a restorable replica

7.1.1.5 ACC Errors

This section contains ndbd error codes that are associated with problems in the ACC (access control and lock management) NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_NO_MORE_INDEX_MEMORY	Resource	Out of index memory during system restart, please increase IndexMemory

7.1.1.6 TUP Errors

This section contains ndbd error codes that are associated with problems in the TUP (tuple management) NDB kernel block.

Error Code	Error Classification	Error Text
NDBD_EXIT_SR_OUT_OF_DATAMEMORY	Out of memory	Out of data memory during system restart, please increase DataMemory

7.1.1.7 LQH Errors

There is currently one ndbd error code associated with the LQH kernel block. This error code was added in NDB 7.2.6, and is shown in the following table:

Error Code	Error Classification	Error Text
NDBD_EXIT_LCP_SCAN_WATCHDOG_FAIL	Fatal	DBLQH fragment scan watchdog detected a problem. Please report a bug.

At the lowest level, an LCP comprises a series of fragment scans. Scans are requested by the DBDIH Master using an LCP_FRAG_ORD signal to the DBLQH kernel block. DBLQH then asks the BACKUP block to perform a scan of the fragment, recording the resulting data to disk. This scan is run through the DBLQH block. See also Section 4.7, “The DBLQH Block”.

7.1.1.8 NDBFS Errors

This section contains ndbd error codes that are associated with problems in the NDBFS (filesystem) NDB kernel block.

Most of these errors will provide additional information, such as operating system error codes, when they are generated.

Error Code	Error Classification	Error Text
NDBD_EXIT_AFS_NOPATH	*XIE*	No file system path
2802	*XIE*	Channel is full
2803	*XIE*	No more threads
NDBD_EXIT_AFS_PARAMETER	*XIE*	Bad parameter
NDBD_EXIT_AFS_INVALIDPATH	*XCE*	Illegal file system path
NDBD_EXIT_AFS_MAXOPEN	*XCR*	Max number of open files exceeded, please increase MaxNoOfOpenFiles
NDBD_EXIT_AFS_ALREADY_OPEN	*XIE*	File has already been opened
NDBD_EXIT_AFS_ENVIRONMENT	*XIE*	Environment error using file
NDBD_EXIT_AFS_TEMP_NO_ACCESS	*XIE*	Temporary on access to file
NDBD_EXIT_AFS_DISK_FULL	*XFD*	The file system is full
NDBD_EXIT_AFS_PERMISSION_DENIED	*XCE*	Read permission denied for file
NDBD_EXIT_AFS_INVALID_PARAM	*XCE*	Invalid parameter for file
NDBD_EXIT_AFS_UNKNOWN	*XIE*	Unknown file system error
NDBD_EXIT_AFS_NO_MORE_RESOURCES	*XIE*	System reports no more file system resources
NDBD_EXIT_AFS_NO_SUCH_FILE	*XFI*	File not found
NDBD_EXIT_AFS_READ_UNDERFLOW	*XFR*	Read underflow
NDBD_EXIT_INVALID_LCP_FILE	*XIE*	Invalid LCP

Error Code	Error Classification	Error Text
NDBD_EXIT_INSUFFICIENT_NODES	*XRE*	Insufficent nodes for system restart
NDBD_EXIT_UNSUPPORTED_VERSION	*XRE*	Unsupported version
NDBD_EXIT_RESTORE_SCHEMA	*XCR*	Failure to restore schema
NDBD_EXIT_GRACEFUL_SHUTDOWN_ERROR	*XCE*	Graceful shutdown not 100% possible due to mixed ndbd versions

7.1.1.9 Sentinel Errors

A special case, to handle unknown or previously unclassified errors. *You should always report a bug using http://bugs.mysql.com/ if you can repeat a problem giving rise to this error consistently*.

Error Code	Error Classification	Error Text
0	*XUE*	No message slogan found (please report a bug if you get this error code)

7.1.2 ndbd Error Classifications

This section lists the classifications for the error messages described in Section 7.1.1, “ndbd Error Codes”.

Error Code	Error Classification	Error Text
XNE	*Success*	No error
XUE	*Unknown*	Unknown
XIE	*XST_R*	Internal error, programming error or missing error message, please report a bug
XCE	*Permanent error, external action needed*	Configuration error
XAE	*Temporary error, restart node*	Arbitration error
XRE	*Temporary error, restart node*	Restart error
XCR	*Permanent error, external action needed*	Resource configuration error
XFF	*Permanent error, external*	File system full

Error Code	Error Classification	Error Text
	action needed	
XFI	*Ndbd file system error, restart node initial*	Ndbd file system inconsistency error, please report a bug
XFL	*Ndbd file system error, restart node initial*	Ndbd file system limit exceeded

7.2 NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

Error Code	Error Name	Error Text
0x00	**TE_NO_ERROR**	No error
0x01	**TE_ERROR_CLOSING_SOCKET**	Error found during closing of socket
0x02	**TE_ERROR_IN_SELECT_BEFORE_ACCEPT**	Error found before accept. The transporter will retry
0x03	**TE_INVALID_MESSAGE_LENGTH**	Error found in message (invalid message length)
0x04	**TE_INVALID_CHECKSUM**	Error found in message (checksum)
0x05	**TE_COULD_NOT_CREATE_SOCKET**	Error found while creating socket(can't create socket)
0x06	**TE_COULD_NOT_BIND_SOCKET**	Error found while binding server socket
0x07	**TE_LISTEN_FAILED**	Error found while listening to server socket
0x08	**TE_ACCEPT_RETURN_ERROR**	Error found during accept(accept return error)
0x0b	**TE_SHM_DISCONNECT**	The remote node has disconnected

Error Code	Error Name	Error Text
0x0c	**TE_SHM_IPC_STAT**	Unable to check shm segment
0x0d	**TE_SHM_UNABLE_TO_CREATE_SEGMENT**	Unable to create shm segment
0x0e	**TE_SHM_UNABLE_TO_ATTACH_SEGMENT**	Unable to attach shm segment
0x0f	**TE_SHM_UNABLE_TO_REMOVE_SEGMENT**	Unable to remove shm segment
0x10	**TE_TOO_SMALL_SIGID**	Sig ID too small
0x11	**TE_TOO_LARGE_SIGID**	Sig ID too large
0x12	**TE_WAIT_STACK_FULL**	Wait stack was full
0x13	**TE_RECEIVE_BUFFER_FULL**	Receive buffer was full
0x14	**TE_SIGNAL_LOST_SEND_BUFFER_FULL**	Send buffer was full,and trying to force send fails
0x15	**TE_SIGNAL_LOST**	Send failed for unknown reason(signal lost)
0x16	**TE_SEND_BUFFER_FULL**	The send buffer was full, but sleeping for a while solved
0x0017	**TE_SCI_LINK_ERROR**	There is no link from this node to the switch
0x18	**TE_SCI_UNABLE_TO_START_SEQUENCE**	Could not start a sequence, because system resources are exumed or no sequence has been created
0x19	**TE_SCI_UNABLE_TO_REMOVE_SEQUENCE**	Could not remove a sequence
0x1a	**TE_SCI_UNABLE_TO_CREATE_SEQUENCE**	Could not create a sequence, because system resources are exempted. Must reboot
0x1b	**TE_SCI_UNRECOVERABLE_DATA_TFX_ERROR**	Tried to send data on redundant link but failed
0x1c	**TE_SCI_CANNOT_INIT_LOCALSEGMENT**	Cannot initialize local segment
0x1d	**TE_SCI_CANNOT_MAP_REMOTESEGMENT**	Cannot map remote segment

Error Code	Error Name	Error Text
0x1e	**TE_SCI_UNABLE_TO_UNMAP_SEGMENT**	Cannot free the resources used by this segment (step 1)
0x1f	**TE_SCI_UNABLE_TO_REMOVE_SEGMENT**	Cannot free the resources used by this segment (step 2)
0x20	**TE_SCI_UNABLE_TO_DISCONNECT_SEGMENT**	Cannot disconnect from a remote segment
0x21	**TE_SHM_IPC_PERMANENT**	Shm ipc Permanent error
0x22	**TE_SCI_UNABLE_TO_CLOSE_CHANNEL**	Unable to close the sci channel and the resources allocated

Appendix A NDB Internals Glossary

This appendix contains terms and abbreviations that are found in or useful to understanding the NDB source code.

ACC. **ACC**elerator or **ACC**ess manager. Handles hash indexes of primary keys, providing fast access to records. See Section 4.3, “The DBACC Block”.

API node. In NDB terms, this is any application that accesses cluster data using the NDB API, including mysqld when functioning as an API node. (MySQL servers acting in this capacity are also referred to as “SQL nodes”.) Sometimes abbreviated informally as “API”. See NDB Cluster Nodes, Node Groups, Replicas, and Partitions.

BACKUP. In the *NDB* kernel, the block having this name performs online backups and checkpoints. For more information, see Section 4.1, “The BACKUP Block”.

CMVMI. Stands for **C**luster **M**anager **V**irtual **M**achine **I**nterface. An NDB kernel handling nonsignal requests to the operating system, as well as configuration management, interaction with the cluster management server, and interaction between various kernel blocks and the NDB virtual machine. See Section 4.2, “The CMVMI Block”, for more information.

CNTR. Stands for restart **C**oordi**N**a**T**o**R**. See Section 4.14, “The NDBCNTR Block”, for more information.

DBINFO. The **D**ata**b**ase **Info**rmation block provides support for the ndbinfo information database used to obtain information about data node internals. See Section 4.6, “The DBINFO Block”.

DBTC. The transaction coordinator (also sometimes written simply as **TC**). See Section 4.9, “The DBTC Block”, for more information.

DICT. The NDB data **DICT**ionary kernel block. Also **DBDICT**. See Section 4.4, “The DBDICT Block”.

DIH. **DI**stribution **H**andler. An NDB kernel block. See Section 4.5, “The DBDIH Block”.

LDM. **L**ocal **D**ata **M**anager. This set of NDB kernel blocks executes the code that manages the data handled on a given data node. It includes the DBTUP, DBACC, DBLQH, DBTUX, BACKUP, TSMAN, LGMAN, PGMAN, and RESTORE blocks.

Each such set of modules is referred to as an LDM instance, and is responsible for tuple storage, hash and T-tree indexes, page buffer and tablespace management, writing and restoring local checkpoints, and Disk Data log management. A data node can have multiple LDM instances, each of which can be distributed among a set of threads. Each LDM instance works with its own partition of the data.

LGMAN. The **L**og **G**roup **MAN**ager NDB kernel block, used for NDB Cluster Disk Data tables. See Section 4.13, “The LGMAN Block”.

LQH. **L**ocal **Q**uery **H**andler. NDB kernel block, discussed in Section 4.7, “The DBLQH Block”.

MGM. **M**ana**G**e**M**ent node (or management server). Implemented as the ndb_mgmd server daemon. Responsible for passing cluster configuration information to data nodes and performing functions such as starting and stopping nodes. Accessed by the user by means of the cluster management client (ndb_mgm). A discussion of management nodes can be found in ndb_mgmd — The NDB Cluster Management Server Daemon.

NDB_STTOR. NDB **ST**ar**T** **O**r **R**estart

QMGR. The cluster management block in the NDB kernel. It responsibilities include monitoring heartbeats from data and API nodes. See Section 4.17, “The QMGR Block”, for more information.

RBR. **R**ow-**B**ased **R**eplication. NDB Cluster Replication is row-based replication. See NDB Cluster Replication.

STTOR. **ST**ar**T** **O**r **R**estart

SUMA. The cluster **SU**bscription **MA**nager. See Section 4.19, “The SUMA Block”.

TC. **T**ransaction **C**oordinator. See Section 4.9, “The DBTC Block”.

TRIX. Stands for **TR**ansactions and **I**nde**X**es, which are managed by the NDB kernel block having this name. See Section 4.23, “The TRIX Block”.

TSMAN. **T**able **s**pace **m**anager. Handles tablespaces for NDB Cluster Disk Data. See Section 4.22, “The TSMAN Block”, for more information.

TUP. **TUP**le. Unit of data storage. Also used (along with **DBTUP**) to refer to the NDB kernel's tuple management block, which is discussed in Section 4.10, “The DBTUP Block”.

Index

D

DUMP commands
  NDB Cluster, 7

E

error messages
  NDB API, 107
errors
  MGM API, 107
  NDB API, 107

M

MGM API
  errors, 107

N

NDB API
  error messages, 107
  errors, 107
NDB Cluster
  DUMP commands, 7
ndb_mgm
  DUMP commands, 7