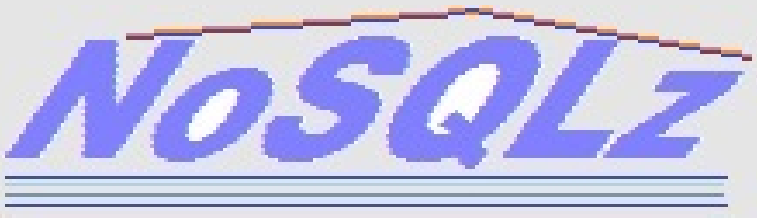

User's Guide

version 1.2.0

NoSQLz, Version 1 Release 2 Level 0

First Edition (March 2014)

This document applies to *NoSQLz* Version 1 Release 2 Level 0.

Disclaimer

NoSQLz version 1 is hereinafter referred to as "this product". The materials provided shall fully be included in all disclaimers of responsibility.

This product contains free software, not to be charged for, except for handling costs.

This product may be freely duplicated.

All disclaimers of responsibility, above and below, apply to all persons and/or installations or any other entity editing, duplicating, processing, or otherwise handling this product. No responsibilities are assumed by any of these persons, installations, corporations, entities, institutions, or organizations.

No warranty, expressed or implied, is provided by any or all of the persons and/or installations editing, duplicating, or otherwise handling this product, as well as by all authors and contributors of material to this product. Persons and/or installations using any of the programs or materials of this product, do so entirely at their own risk.

No warranty is made to the accuracy of the programs or related material and no responsibility is assumed for any modification directly or indirectly caused by the use of this software. It is the user's responsibility to evaluate the usefulness of the material supplied.

We do not guarantee to keep any material provided up to date, nor do we guarantee to provide any corrections or extensions described by anyone, or corrections or extensions made in the future.

No warranty, expressed or implied, is provided by any or all of the persons and/or installations editing, duplicating, or otherwise handling this product, as well as by all authors and contributors of material to this product. Persons and/or installations using any of the programs or materials of this product, do so entirely at their own risk.

The editor and/or any other persons, institutions, corporations, or any other entity handling this product, as well as any authors or contributors of material to this product, do not assume any responsibility, express, implied, or to be construed in any way, to update this product at all.


NoSQLz

Contents

1. Introduction

NoSQLz is a key-value Database Management System running under z/OS. It provides direct or sequential access to schema-less tables.

A table dataset (or database) is a VSAM dataset that resides on one (or several) disks.

Table data consists of 'format-agnostic' variable-length records. Records are handled through create, read, update and delete functions.

Record size goes from 500 bytes (minimum) to 75 MB (large records supported). There may be up to approximately 1,000,000,000 records in a single table.

A record key is always a variable-length string, from 1 to 250 bytes. Records may have different key-lengths in the same database. For this reason, key length must always be supplied to access records. A record is uniquely referred to by the combination (key length, key value).

ACID properties are provided so that you may write application programs that implement database transactions. A transaction is a set of information processing operations that leave the database in a consistent state: the operations that a transaction consists of are either all completed successfully or all cancelled successfully. The COMMIT processing takes care of consistency to maintain ACID properties throughout the life of the databases.

With NoSQLz, data retrieval is always favored, at the possible expense of updates.

This free version of NoSQLz is destined to a monoplex. A chargeable version provides multiple concurrent accesses natively throughout the sysplex.

NoSQLz may be used for highly-available applications, data warehousing, document store, Internet applications, OLAP, etc.


NoSQLz

2. Installation

2.1 System requirements

You must run a recent version of z/OS.

CPACF hash functions must be active (at least SHA-1 must be supported by the hardware).

This version only supports a monoplex system.

2.2 Installation steps

2.2.1 Uploading product libraries

The binary file (for example: NSQ.NSQINST.XMI) must be transferred to MVS into a sequential LRECL 80 file.

Using the binary transfer method, upload it to your mainframe host into a pre-allocated, LRECL 80, sequential dataset.

You must now enter the following TSO command against the file:

```
RECEIVE INDA('NSQ.NSQINST.XMI')
```

Respond as follows to the "Enter restore parameters…." message:

```
DA('desired.installation.dataset.name')
```

or :

```
DA('desired.installation.dataset.name') VOL(desired.allocation.volume)
```

Adapt and submit member **RECEIVE** of the newly created partitioned dataset. JCL variable INSTDS refers to the PDS you have just created using the RECEIVE command. JCL variable PFXC contain the high level qualifiers to assign to the product libraries.

This will create and upload all libraries from the installation members. You should not change the last dsn qualifier of the libraries (for example: NSQCNTL).

List of libraries:

Last dsn qualifier	Usage
NSQLOAD	Load library
NSQCNTL	JCLs and parameters

NSQTEST	Samples

2.2.2 APF authorization

You have to specify in the PROGxx member of the current PARMLIB the dsname of the NoSQLz load-module library, and the name of the disk volume it resides on, for example:

```
     ...
APF ADD  DSNAME(PROD.NSQ.V120.NSQLOAD)  VOLUME(VOL001)
     ...
```

Make sure the new entry is taken into account by the system: this requires an IPL, or a SET PROG=xx command, or the SETPROG command:

```
SETPROG APF,ADD,DSN=PROD.NSQ.V120.NSQLOAD,VOL=VOL001
```

An alternate option consists in copying NoSQLz authorized modules to a system library or to an existing APF library. The modules that must be copied are the ones that are marked AC=1.

2.2.3 Link-list (optional)

If the NoSQLz product becomes heavily used, it may be preferable to put the NoSQLz load library in link-list. The LNKLSTxx member of the system PARMLIB should be updated accordingly.

The alternative is to use the NoSQLz load library as STEPLIB or JOBLIB.

2.2.4 Prepare the start procedure

NoSQLz necessitates that a **NSQTASK** started task be activated.

Copy the member NSQCNTL(NSQTASK) in one of your system PROCLIBs. Set variable NSQLOAD to the name of the product load library. DD NSQPARM may point to any PDS you wish. NSQPARM is discussed below in paragraph **Set NoSQLz parameters**.

Have the started task NSQTASK launched just after IPL.

The P= parameter of the NSQTASK procedure stands for the name of your NoSQLz system. It should be a 4-byte name, for example TEST, or PRD1.

The START command should be in the following form:

```
S NSQTASK,REUSASID=YES[,P=<NoSQLz system name]
```

The STEPLIB must be APF-authorized (see paragraph « APF authorization »).

The RACF userid associated with the NSQTASK STC must be authorized in UPDATE to all databases that may be handled by the NoSQLz system.

2.2.5 Prepare the COMMIT dataset.

The COMMIT dataset (or COMDS) is required for COMMIT processing. This dataset can be allocated and formatted using the sample JCL in NSQCNTL(COMDSINI).

2.2.6 Set NoSQLz parameters

The NSQPARM DD statement of the NSQTASK procedure refers to a PDS member that will contain all NoSQLz parameters ('NSQPARM parameters'). It is recommended to set the 4-byte NoSQLz system name as suffix of the member name, for example NSQP**PRD1** if the NoSQLz system name is **PRD1**.

The member should at least contain a statement to describe the required COMMIT dataset:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+

DBNAME   NSQCOMDS PROD1.COMDS                                    W
```

A sample may be found in member NSQCNTL(NSQPARM).

2.2.7 Start the procedure

The **NSQTASK** started task (or whatever name you gave it) may now be activated.

2.2.8 Verification procedure

You may want to use the provided samples to test that the installation is OK. The NSQTASK started task should not be active at this point.

1. JCL DBSTATES of NSQ.NSQTEST must be adapted and submitted to initialize and load a DBSTATES NoSQLz database. This JCL will create a DBSTATES table containing all states of the USA.

2. JCL DBCITIES of NSQ.NSQTEST must be adapted and submitted to initialize and load a DBCITIES NoSQLz database. You must manually replace NSQ.NSQTEST by the name of your NSQTEST library (in step 'PREPARE'). This JCL will create a DBCITIES table containing all cities and towns of the USA.

The ouput message IEC161I is normal in steps 1 and 2.

3. You must check that the NSQPARM parameters contain a statement for both DBSTATES and DBCITIES (in addition to NSQCOMDS), for example :

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+

DBNAME   DBSTATES PRD1.DBCITIES                                  W

DBNAME   DBSTATES PRD1.DBSTATES                                  W
```

4. The NSQTASK procedure may now be started. The NoSQLz system name may be set (for example) to PRD1 and NSQPARM must point to the parameters described in step 3.

5. You may run the REXXCITY REXX exec under TSO. This EXEC can be found in the NSQTEST library. It prompts you for a US city and, by requests to the PRD1 NoSQLz system, displays all towns or cities of that name that can be found in the DBCITIES database.

For example:

```
TSO EX 'NSQ.NSQTEST(REXXCITY)'

51 states read

 Enter the name of a state / city / town / village in the USA:

new york

Searching for:  New York

 New York township              Missouri            00000  Population=269

 New York                       New York            00000  Population=19465197

 New York city                  New York            51000  Population=8244910

 New York County                New York            00000  Population=1601948

 ***
```

3. Database management

3.1 Creating a database

To create a database, you need to fix the following parameters:

- the name of the database (8-byte name): this name will serve to uniquely identify the database in the NoSQLz system;
- the average size of a record (key + data);
- the maximum number of records to be stored in the database.

The database is tailored, allocated on disk, and formatted (using the NSQDUT01 utility). Once formatted, it cannot take new extents: it is built for the maximum number of records that has been set (this number is actually rounded up to the next power of 2, so it can be larger than you may think). The average record size is always rounded up to a VSAM CI size (control interval size), because only one record is stored in a CI. You must keep that in mind when you plan to allocate a new database and understand the implications for your site as far as DASD storage is concerned.

A sample JCL may be found in NSQCNTL(DBINIT).

3.2 Declaring a database

Each database must be declared in the NSQPARM parameters by a DBNAME statement:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+

DBNAME   NSQCOMDS PROD1.DBCITIES                                 W
```

The DBNAME statement consists of:

- the NoSQLz name of the database: up to 8 bytes, from column 10;
- the dsname of the associated VSAM file: up to 44 bytes, from column 19;
- the access option, that sets whether the database is accessed in read/write ('W') or in read-only ('R'): one byte at column 64.

See also sample statements in member NSQCNTL(NSQPARM).

3.3 Database clean-up

Databases never need reorganization. Because NoSQLz is based on timestamp-based concurrency control and multiversion concurrency control, databases need frequent clean-ups, using the **NSQDUT04** utility. The clean-up process will suppress the following entries in the databases:

- uncommitted records (records created but never committed)
- outdated records (old versions of records, no longer referenced in the database index)
- logically deleted records (DELETE function call)
- malformed records or records in error

A sample JCL may be found in NSQCNTL(DBCLEAN).

3.4 Database initial load

A database may be initially loaded by an application program with a series of **CREATE** function calls, or by using the **NSQDUT02** batch utility. It may also remain empty until online CREATE function calls populate it.

3.5 Database security

As for any dataset, access to databases is controlled based on the RACF userid that the access originates from. An open error will happen if the user is not authorized to access the dataset.

4. Function calls

4.1 Description

All function calls always require a "parameter block" (NSQ_BLOCK) as first parameter to specify the table involved, the key value, etc. This block is described in the **Appendix**. You must test the return code (NSQ_RETCD): a value of 0 (or 4) means the operation completed successfully (or partly).

There may also be a second parameter to be provided, to obtain or write record data.

To invoke NoSQLz services, you must call the "transaction call API routine" **NSQTCAPI**. that is provide in the NoSQLz LOAD library.

Example in Cobol (with 2 parameters):

```
CALL 'NSQTCAPI' USING NSQ-BLOCK, OUTZONE.
```

Example in assembler (with 2 parameters):

```
CALL NSQTCAPI,(NSQ_BLOCK,OUTZONE),VL
```

Example in REXX (with 2 parameters):

```
ADDRESS LINKPGM "NSQTCAPI NSQ_BLOCK OUTZONE"
```

Several NoSQLz systems may be active in your system. You have to specify which system your application program requires access to. This can be done by specifying in your JCL a DD statement in the following form:

```
//NSQLxxxx  DD   DUMMY
```

where "xxxx" stands for the name of the NoSQLz system. If you use REXX, you may also code a REXX statement as follows:

```
ADDRESS TSO  "ALLOC F(NSQLxxxx) DUMMY"
```

The basic element of processing is the task: a task can run only one transaction at a time. The transaction begins by any NoSQLz function call, and terminates either by a COMMIT function call or a ROLLBACK function call. After that, the task may start a new transaction or stop.

4.2 The READ function

The READ function is used to directly access a record by key.

The returned record is always 'older' than the transaction. You cannot read records created after the time your transaction began, or records currently modified by another transaction.

Parameters: NSQ_BLOCK and output zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READ ' (8-byte literal, left-justified)
NSQ_OPT	'N' (1-byte literal) to get the lastest version of the record
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to search (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided output zone (4-byte integer)

'O' in the NSQ_OPT field can be used to retrieve the oldest version of the record, if it has not been deleted by the clean-up utility.

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)
NSQ_OUTLN	Number of bytes of data returned in the provided output zone (4-byte integer)

NSQ_RETCD will contain 0 if a whole record has been returned. NSQ_RETCD will contain 4 if the record has been truncated (output zone was too small, or the whole record could not be read).

NSQ_RETCD will contain 10 in the unlikely case that the record has been modified at least 2 times since your transaction started. You should restart the transaction.

Other values of NSQ_RETCD mean that no record matches the provided key, or that an error occurred.

Output (output zone): will contain the data part of the record, if the NSQ_RETCD returned is 0 or 4.

Samples: NSQTEST(ASMREAD), NSQTEST(COBREAD), NSQTEST(REXREAD).

4.3 The READSEQ function

The READSEQ function is used to sequentially read database records. Each READSEQ call returns one record, in no specific key order. The record key is returned in the NSQ_BLOCK while the data is returned in the output zone.

READSEQ does not necessarily provide a consistent view of the data, especially when the update rate is high. Records that are being updated and new records are not returned. You can never sure that you got all records of the database through one READSEQ loop.

Parameters: NSQ_BLOCK and output zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READSEQ ' (8-byte literal, left-justified)
NSQ_OPT	'N' (1-byte literal) to get the lastest version of the record
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_INLEN	Length of the provided output zone (4-byte integer)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)
NSQ_KYLEN	Length of key of returned record (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key of returned record (length is NSQ_KYLEN bytes)
NSQ_OUTLN	Number of bytes of data returned in the provided output zone (4-byte integer)

NSQ_RETCD will contain 0 if a whole record has been returned. NSQ_RETCD will contain 4 if the record has been truncated (output zone was too small, or the whole record could not be read). NSQ_RETCD will contain 14 when no more record can be returned because the end of the database has been reached.

Other values of NSQ_RETCD mean that an error occurred.

Output (output zone): will contain the data part of the record, if the NSQ_RETCD returned is 0 or 4.

Samples: NSQTEST(COBREADS), NSQTEST(REXREADS).

4.4 The READTEST function

The READTEST function is used to test whether a specify key exists in the database (and matches an existing record).

No record data is returned by this function.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'READTEST' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to search (length in bytes from 1 to NSQ_KYLEN)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the specified key matches an existing record.

The NSQ_MESSG zone will display some technical information about the index entry of the key.

Other values of NSQ_RETCD mean that no record matches the provided key, or that an error occurred.

Samples: NSQTEST(COBREADT), NSQTEST(REXREADT).

4.5 The CREATE function

The CREATE function is used to create a new record in the database under a new key (the key does not match any existing record). The created record will not be visible to any user (except your application program) until the COMMIT function is called.

Parameters: NSQ_BLOCK and input zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'CREATE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to create (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided input zone (4-byte integer)

Input (input zone): contains the data part of the record to be created.

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been created. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the record will be not retrieved again.

NSQ_RETCD will contain 9 if an existing record with the same key already exists.

NSQ_RETCD will contain 24 if no room has been found to insert the new record. Either the database is full, or too many uncommitted records with the same key have been created. In either case, the database should be cleaned up using the NSQDUT04 utility.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBCREAT), NSQTEST(REXCREAT).

4.6 The UPDATE function

The UPDATE function is used to update an existing record. The updated record will not be visible to any user (except your application program) until the COMMIT function is called.

It need not be preceded by a READ function call, although it may generally be advisable.

Parameters: NSQ_BLOCK and input zone.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)

NSQ_ACT	'UPDATE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to update (length in bytes from 1 to NSQ_KYLEN)
NSQ_INLEN	Length of the provided input zone (4-byte integer)

Input (input zone): contains the data part of the record to be updated.

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been updated. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the previous version of the record will stay active.

NSQ_RETCD will contain 8 if the record has not been found.

NSQ_RETCD will contain 24 if no room has been found to update the record. Either the database is full, or too many uncommitted records with the same key have been created. In either case, the database should be cleaned up using the NSQDUT04 utility.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBUPDAT), NSQTEST(REXUPDAT).

4.7 The DELETE function

The DELETE function is used to delete a specific key from the database. The effect of the delete operation will not be visible to any user (except your application program) until the COMMIT function is called.

Note that it is a logical delete, the matching record will be deleted physically by the NSQDUT04 clean-up utility. A READ function call with NSQ_OPT = 'N' (newest) will return an error, whereas a READ with NSQ_OPT = 'O' (oldest) will return the deleted record.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'DELETE ' (8-byte literal, left-justified)
NSQ_TBLNM	Name of database (8 bytes, left-justified)
NSQ_KYLEN	Length of key (4-byte integer, value is 1 to 250)
NSQ_KYVAL	Value of key for record to update (length in bytes from 1 to NSQ_KYLEN)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the record has been deleted. Note that a subsequent COMMIT function call must be issued, otherwise the index of the database will not be updated and the current version of the record will stay active.

NSQ_RETCD will contain 8 if the record has not been found.

Other values of NSQ_RETCD mean that an error occurred.

Samples: NSQTEST(COBDELET), NSQTEST(REXDELET).

4.8 The COMMIT function

The COMMIT function is used to end the transaction and to make the database updates (resulting from CREATE, UPDATE or DELETE function calls) visible to all users. It is assumed that your transaction leaves the databases in a consistent state between two commits.

Database positioning is not lost after COMMIT: for example, READSEQ may continue as if no COMMIT took place.

If your transaction does not end with a COMMIT call, any update previously done by a CREATE, UPDATE or DELETE function call will be ignored (actually, it will stay uncommitted and the next database clean-up will erase it).

The COMMIT process can only handle a limited number of updates, so you should have an online transaction generate only a few updates before committing.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'COMMIT ' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Error message (100 bytes)

NSQ_RETCD will contain 0 if the COMMIT call has been successful.

Other values of NSQ_RETCD mean that an error occurred. For example, there may be a concurrent access to a record key, or a record may have been modified after your transaction started. A COMMIT call is by no means assured of success, because of concurrent access.

4.9 The ROLLBACK function

The ROLLBACK function is used to end the transaction and to discard the database updates that it may have done.

Database positioning is not lost after ROLLBACK: for example, READSEQ may continue as if no ROLLBACK took place.

With the ROLLBACK function, any update previously done by a CREATE, UPDATE or DELETE function call is ignored (actually, it stays uncommitted and the next database clean-up will erase it).

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'ROLLBACK' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Information message (100 bytes)

NSQ_RETCD will always contain 0.

4.10 The FREEALL function

The FREEALL function is used to free all databases that your task may have used till now. This function call should be used with care, because the FREEALL function call will close and free all databases that has been used by the current task. FREEALL may prove useful in a TSO/REXX environment, at the end of the REXX exec: the databases will not remain allocated to your TSO user.

Parameters: NSQ_BLOCK only.

Input data (NSQ_BLOCK):

NSQ_VRSN	'1' (1-byte literal)
NSQ_ACT	'FREEALL' (8-byte literal, left-justified)

Output data (NSQ_BLOCK):

NSQ_RETCD	Return code (4-byte integer)
NSQ_MESSG	Information message (100 bytes)

NSQ_RETCD will always contain 0.

5. Managing the NoSQLz started task

5.1 NSQPARM parameters

The NSQPARM DD statement of the NSQTASK procedure points to a PDS member that contains all NoSQLz parameters ('NSQPARM parameters').

It is recommended to set the 4-byte NoSQLz system name as suffix of the member name, for example NSQP**PRD1** if the NoSQLz system name is **PRD1**.

5.1.1 DBNAME

The DBNAME parameter is used to declare a NoSQLz database. See paragraph "**Declaring a database**" above.

5.1.2 COMTIME

The COMTIME parameter is used to set the maximum time application programs will wait for one COMMIT process to complete. When this time is exceeded, the COMMIT ends with a NSQ_RETCD return code 13 (NSQ_MESSG="NO ANSWER FROM COMMIT SUBTASK").

The COMTIME statement consists of:

- the maximum duration set for the COMMIT process, expressed in hundredths of a second: up to 10 numeric bytes, from column 10.

Example:

```
*---+----1----+----2----+----3----+----4----+----5----+----6----+----

COMTIME  1000                    MAX DURATION ALLOWED FOR COMMIT: 10 SEC.
```

5.2 Starting and stopping the started task

The member NSQCNTL(NSQTASK) should be copied into one of your system PROCLIBs.

The START command should be in the following form:

```
S NSQTASK,REUSASID=YES[,P=<NoSQLz system name]
```

The P= parameter stands for the name of your NoSQLz system. It should be a 4-byte name, for example TEST, or PRD1. Applications programs refer to this NoSQLz system name through a specific DD statement, for example:

```
//NSQLPRD1  DD   DUMMY  (for invoking the PRD1 NoSQLz system)
```

You may want to rename the NSQTASK to a name suffixed by the NoSQLz system name, for example NSQPRD1 if the NoSQLz system name is PRD1.

To stop the NoSQLz started task, you make use of the MVS STOP (P) command, for example:

```
P NSQTASK
```

Function calls will not work when the NoSQLz started task is inactive.

5.3 Commands to the started task

The MVS MODIFY (F) command can be used to display information or to modify the status of a database.

5.3.1 Display databases

```
F NSQTASK,LISTDB
```

This command will list all the databases declared to the NoSQLz system. For each database, the command displays: its dbname, access mode (R or W), dsname and the number of committed updates to this database.

```
f nsqtask,listdb

     NOSQLZ PRD1 DBCITIES W PROD.DBCITIES     0000000000

     NOSQLZ PRD1 DBCITIE2 W PROD.DBCITIE2     0000000051

     NOSQLZ PRD1 DBSTATES W PROD.DBSTATES     0000000000

     NOSQLZ PRD1 NSQCOMDS W PROD.COMDS        0000000000
```

5.3.2 Display current commit

```
F NSQTASK,LISTCC
```

This command will list the number of commits that have been achieved. If a commit is underway, it will display the time and date of the commit, and the involved job.

Example:

```
f nsqtask,listcc

     NOSQLZ PRD1 NUMBER OF COMMITS: 0000000001

     NOSQLZ PRD1 - NO COMMIT BEING DONE CURRENTLY

     NOSQLZ PRD1 - LAST COMMIT DONE FOR PROD02CC

f nsqtask,listcc

     NOSQLZ PRD1 NUMBER OF COMMITS: 0000000002
```

```
NOSQLZ PRD1 COMMIT TIME=14090138 DATE=20140331 TOD=00CCEF7E0A085AD0

NOSQLZ PRD1 FOR JOBNAME=PRODUPDT ASID=0039 (HEX)
```

5.3.3 Display tasks

```
F NSQTASK,LISTTSK
```

This command will list the subtasks of the started task, such as the commit task.

5.3.4 Display log entries

```
F NSQTASK,LISTLOG
```

This command will list the more recent log entries, describing the recent events: start of transaction, transaction commit and end of transaction. "S=" displays the transaction start time, "C=" displays either the transaction commit time or the transaction end time, in tod clock format.

Example:

```
f nsqtask,listlog

     NOSQLZ PRD1 START  PROD02T  S=CCEF7AE2D01270 13:54:54 C=00000000000000

     NOSQLZ PRD1 START  PROD01CC S=CCEF7B98BC5E30 13:58:05 C=00000000000000

     NOSQLZ PRD1 COMMIT PROD01CC S=CCEF7B98BC5E30 13:58:09 C=CCEF7B9BF59C50

     NOSQLZ PRD1 END000 PROD01CC S=CCEF7B98BC5E30 13:58:09 C=CCEF7B9C0B02E0
```

5.3.5 Display communication area

```
F NSQTASK,LISTDATA
```

This command will list the NoSQLz communication area. It may be used for debugging.

5.3.6 Close and free a database

```
F NSQTASK,F <dbname>
```

This command will close and free the database whose 8-byte dbname is mentioned. It must be used if you want to delete or to rebuild the database. The NoSQLz LISTDB command will display the database with a “X” tag, meaning it is no longer available to application programs.

Example:

```
f nsqtask,f dbcities

     MAIN - DATABASE DBCITIES HAS BEEN FREED

f nsqtask,listdb
```

```
NOSQLZ PRD1 DBCITIES W PROD.DBCITIES     0000000000  X

NOSQLZ PRD1 NSQCOMDS W PROD.COMDS        0000000000
```

5.3.7 Allocate and open a database

F NSQTASK,A <dbname>

This command will allocate and open the database whose 8-byte dbname is mentioned. The database must have been freed by a previous F command. This command is required if you want to put the database in use again. The database must have been previously declared in the NSQPARM parameters. You cannot put in use a database that has not been declared.

Example:

```
f nsqtask,a dbcities

     DBAS - ALLOCATING DBCITIES DSN=PROD.DBCITIES

     MAIN - DATABASE DBCITIES READY, DSN=PROD.DBCITIES
```

6. Batch utilities

Several batch utilies help you manage databases. All batch utilites run "offline", i.e. they don't need that the NoSQLz started task be active.

All input statements of batch utilities must being at column 1.

6.1 NSQDUT01: database formatting

This utility formats a new NoSQLz database. The database must have been previously created and must be empty.

The occurrence of a IEC161I message during the process is normal.

6.1.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQOUT

Mandatory. Specifies the dataset to be formatted. It must have been previously allocated by a DEFINE CLUSTER IDCAMS command.

6.1.2 Control statements

TBLNAME=<name of database>

Mandatory. Specifies the 8-byte name of the database.

HASHALG=n

Optional. Specifies the hash algorithm chosen for the database. Three values are supported:

1: SHA-1 (default)

2: SHA-256

3: SHA-512

KEYHASHMASK=<8-byte hexadecimal value>

DATAHASHMASK=<8-byte hexadecimal value>

Mandatory. These parameters specify hash masks to apply for the index part and the data part of the database. Values depend on the number of records and the record size. You should let the ZCOMP exec set those parameters (see the sample JCL).

6.1.3 Sample JCL

See NSQCNTL(DBINIT).

6.2 NSQDUT02: database initial loading

This utility loads data into a NoSQLz database that has previously been formatted by the NSQDUT01 utility. The database needs not be empty.

6.2.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQOUT

Mandatory. Specifies the NoSQLz dataset to receive data. It must have been previously formatted by NSQDUT01.

This database should not be under the control of the NoSQLz started task, otherwise concurrent updating may take place and corrupt the data.

SYSUT1

Mandatory. Specifies the input sequential dataset that contains data to load into the NoSQLz dataset.

SYSUTERR

Optional. Specifies an output sequential dataset that will receive SYSUT1 records that could not be inserted in the NSQOUT dataset. This enables you to try to load again the remaining records.

6.2.2 Control statements

KEYLEN=<key size>

Size of the key part of the record. This size is assumed to be fixed, and the key is assumed to be located at start of record. The key length must be from 1 to 250.

DATALEN=<data size>

Size of the data part of the record. This size is assumed to be fixed, and the key is assumed to be located at start of record. The data length must be from 1 to 32760.

DUP=INSERT/IGNORE

This option specifies whether duplicate keys are to be ignored (DUP=IGNORE) or to be inserted as new versions to override existing keys (DUP=INSERT). Default is INSERT.

6.2.3 Sample JCL

See NSQCNTL(DBLOAD).

6.3 NSQDUT03: printing of database statistics

This utility reads an existing database to provide varied statistics. Statistics are written in the syslog via WTOs. For example:

```
+DUT03 - TBL=DBCITIES INDEX-MASK=0000FFFF DATA-MASK=0001FFFF HSHALGO=03
+**  DATA SLOT NUMBER RANGE: 0000065538 - 0000196609
+**   DATABASE RECORDS READ: 0000076120
+**         - INDEX RECORDS: 0000032049 / 0000065536  48%
+**         -  DATA RECORDS: 0000044069 / 0000131071  33%
+** MAX NUMBER OF KEY CELLS: 006 / 017 IN SLOT: 0000007618
+**    MAX DATA SLOT NUMBER: 0000196601
+**  HIGHEST VERSION NUMBER: 001
+**  HIGHEST VALUE FOR SALT: 07 (HEX)
+**        OLDEST TIMESTAMP: 20140331 07375243
+**        LATEST TIMESTAMP: 20140331 07422695
```

This is especially of interest to get the occupation rate of the database (index part and data part).

6.3.1 DD statements

NSQIN

Mandatory. Specifies the NoSQLz dataset to process.

6.3.2 Sample JCL

See NSQCNTL(DBSTATS).

6.4 NSQDUT04: database clean-up

This utility cleans up an existing database to automatically suppress unneeded records and keep only the last version of the records. It deletes uncommitted data and outdated data (except for recent records, depending on the LIMIT value). It also displays final statistics, for example:

```
+DUT04 - DATA RECORDS READ      :0000125811
+DUT04 - DATA RECORDS DELETED:0000081754
+DUT04 -              (NO INDEX ENTRY:0000000002)
+DUT04 -                        (OUTDATED:0000081742)
+DUT04 -        (LOGICALLY DELETED:0000000010)
```

6.4.1 DD statements

SYSIN

Optional. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to process.

6.4.2 Control statements (optional)

SIMULATE=Y/N

Specifies whether physical deletes are carried on or not. Default is N (deletes are physically done).

LIMIT=<number of hours>

The utility must not delete the records that have been created less than <number of hours> ago. This number must be from 0 to 10000. Default is 1 (one hour).

With LIMIT=0, all unneeded records will be suppressed, no matter when they were created. This is in general not advisable, because uncommitted records may be suppressed just before the transaction COMMIT takes place.

6.4.3 Sample JCL

See NSQCNTL(DBCLEAN).

6.5 NSQDUT10: database record printing (from key)

This utility reads a record from a database, based on a key value. It reads both the more recent version of the record, and the oldest, should it exist.

6.5.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to read data from.

6.5.2 Control statements

KEYLEN=<key size>

Size of the record key specified in the V= statement. It must be from 1 to 250.

A value of 0 has a special meaning: it means you want to read all records sequentially. Because it is a physical read, even outdated or uncommitted records are accessed and displayed, so duplicate keys may be listed if no clean-up has been done in the meantime. KEYLEN=0 should be used with care, since it may result in a large amount of data being displayed. For example:

```
//READ1    EXEC PGM=NSQDUT10
//NSQIN    DD   DISP=SHR,DSN=PROD.DBSTATES
//SYSIN    DD   *
KEYLEN=0
```

Results in the following display:

```
+DUT10 - 0000000182 V=01 S=00 KL=075 K=Georgia
+DUT10 - 0000000187 V=01 S=01 KL=075 K=Vermont
+DUT10 - 0000000190 V=01 S=01 KL=075 K=Idaho
```

V=<key value>

Value of the key for the record to be searched. Up to 78 bytes of key may be specified.

DELETE=Y/N

Specifies whether the record that matches the key must be logically deleted or not (default: N).

6.5.3 Sample JCL

See NSQCNTL(RECREAD).

6.6 NSQDUT11: database record printing (from record-id)

This utility reads a record from a database, based on its record-id. It displays the index entry that matches the record-id. It is equivalent to a READTEST function call.

6.6.1 DD statements

SYSIN

Mandatory. Specifies the input dataset that contains the control statements.

NSQIN

Mandatory. Specifies the NoSQLz dataset to read data from.

6.6.2 Control statements

RECID=<record_id (26 hexadecimal digits)>

6.6.3 Sample JCL

See NSQCNTL(RECRBYID).

7. Appendix

7.1 The parameter block

The parameter block must be provided for each function call.

7.1.1 Cobol description

Here is an example for a parameter block whereby a table named MYTABLE is accessed to retrieve ('READ') a record whose key is 'MYKEY' (length 5 bytes). The newest ('N') version of the record is required.

```
01  NSQ-BLOCK.
    02 NSQ-VRSN             PIC X    VALUE '1'.
    02 NSQ-OPT              PIC X    VALUE 'N'.
    02 NSQ-ACT              PIC X(8) VALUE 'READ'.
    02 NSQ-TBLNM            PIC X(8) VALUE 'MYTABLE'.
    02 NSQ-INLEN            PIC S9(5) COMP VALUE 100.
    02 NSQ-OUTLN            PIC S9(5) COMP.
    02 NSQ-RETCD            PIC S9(5) COMP VALUE 0.
    02 NSQ-MESSG            PIC X(100) VALUE SPACES.
    02 NSQ-KYLEN            PIC S9(2) COMP VALUE 5.
    02 NSQ-KYVAL.
       05 NSQ-KYVAL05       PIC X(5) VALUE 'MYKEY'.
       05 FILLER            PIC X(245).
```

7.1.2 Assembler description

```
NSQ_BLOCK EQU  *
NSQ_VRSN  DS   CL1          VERSION OF PARAMETER BLOCK: C'1'
NSQ_OPT   DS   CL1          READ OPTION: 'N'EWEST (RECOMMENDED)
NSQ_ACT   DS   CL8       ACTION ("START", "CREATE", "READ", "UPDATE",
*                                "DELETE", "COMMIT", "ROLLBACK")
NSQ_TBLNM DS   CL8          NAME OF TABLE DATA SET TO BE ACCESSED
* DATA ZONE INFORMATION
NSQ_INLEN DS   CL4   IN     LENGTH OF DATA ZONE SUPPLIED BY CALLER
*                           (FOR ALL OPERATIONS)
NSQ_OUTLN DS   CL4   OUT    LENGTH OF DATA COPIED INTO ZONE BY NOSQLZ
*                           (ONLY FOR "READ" OPERATIONS)
* ERROR INFORMATION
NSQ_RETCD DS   CL4          RETURN CODE
NSQ_MESSG DS   CL100        ERROR MESSAGE (IF RETURN CODE NOT ZERO)
* KEY PARAMETER
NSQ_KYLEN DS   CL2          LENGTH OF KEY (1-250)
NSQ_KYVAL DC   XL250'00'    VALUE OF KEY
```

7.2 Return codes

Here is a list of return codes you may find in field NSQ_RETCD. Additional information is also provided in the NSQ_MESSG field.

Code (decimal)	Code (hexadecimal)	Meaning
0	00	Operation successful
4	04	Record partially read (READ-type operation)
8	08	Record not found
9	09	Record already exists, cannot be created
10	0A	Record not accessible, being updated
12	0C	Record has been logically deleted
13	0D	COMMIT error, cannot modify/delete record
14	0E	End of file (sequential processing)
15	0F	READ option is in error
16	10	VSAM error when accessing database record
20	14	Key length passed is zero or > 250
22	16	NOSQLZ system not found or not active
23	17	Database not found (no identify point)
24	18	Database is full, cannot insert index / data record
25	19	Database in read-only, write not allowed
27	1B	Cannot get CTRL info (hash) from db
28	1C	Cannot get dsname of db (undefined db)
29	1D	Database name error, dbname differs from tbname
30	1E	Name/token create or retrieve error
32	20	Routine error: NSQTENTY routine not found, or parameter block version error, or parameter block action unknown, or parameter block

		action unsupported, or transaction commit time not set
34	22	Database error: incorrect module (OODAT), or allocation error, or open error
36	24	Zone passed by caller is unusable
40	28	Started task NOSQLZ missing

8. Readers' Comments — We'd Like to Hear from You

NoSQLz - User's Guide, 2014 edition

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses.

Name _____ Address _____

Company or Organization _____

Phone No. _____