1. **NKV Storage APIs**

**1.1 Overview**

The following APIs define the core data structures for NKV. User can see NKV as a big pool of network KV storage. This pool of storage can be divided into several logical partitions (pools) or it can be accessed as pass-through mode where individual target nodes (and the physical pool of kv devices hosted on the node) will be exposed to the user of NKV. In non-pass-through mode, NKV will take care of advanced cluster features like intelligent data-placement by supporting logical pools, data redundancy (like replication, EC), efficient data recovery in case of failures and rebalancing of data in case of node addition. But, in case of pass-through mode, Application using NKV needs to take care of all those. NKV will not be doing much other than propagating the errors/failures to the application.

* 1. **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Date | Author | Change |
| 0.1 | 11/23/2018 | Somnath Roy | Created |
| 0.2 | 11/27/2018 | Somnath Roy | Added configuration apis |
| 0.3 | 12/1/2018 | Somnath Roy | Added IO APIs |
| 0.4 | 12/12/2018 | Somnath Roy | Added batch and lock apis |
| 0.5 | 12/20/2018 | Somnath Roy | Added event , indexing and versioning APIs |

* 1. **Initialization and Configuration APIs**

Following are the APIs for NKV initialization and configuration.

* + 1. **nkv\_open**

**int32\_t nkv\_open (const char** \***config\_file, const char\* app\_uuid, const char\* host\_name\_ip; uint32\_t host\_port, uint64\_t \*instance\_uuid, uint64\_t\* nkv\_handle)**

This API initializes the NKV client side data structures by gathering NKV target side information after communicating with NKV-FM. Assumption is, NKV-FM should be up and running before NKV-client is initiating this request. This call will also register the nkv\_handle generated by NKV and the (host name, instance\_uuid) combination to the internal NKV metadata structure to keep track of all NKV instances running presently.

**REQUIRED**

Yes.

**PARAMETERS**

IN **config\_file** -- Absolute path of NKV config file consisting of NKV-FM information and other

configuration related to NKV.

IN **app\_uuid** -- Unique app identifier to support multiple app at the same time on top of NKV.

With this appuuid, NKV generates a unique nkv\_handle to isolate and protect

different app data.

IN **host\_name\_ip –** Host name or IP where NKV instance will be running

IN **host\_port –** Host port where this instance will be running

IN **instance\_uuid --** Unique NKV instance identifier. Could be generated in combination with

(app\_uuid, host\_name\_ip, host\_port)

OUT **nkv\_handle** – A positive unique id for combination of nkv and the application(**not instance**). -1 in case of error.

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_UNREACHABLE – Not able to communicate with NKV-FM

NKV\_ERR\_NULL\_INPUT -- config\_file or app\_uuid is NULL

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_get\_instance\_info**

**int32\_t nkv\_get\_instance\_info (uint64\_t nkv\_handle, uint64\_t instance\_uuid, nkv\_instance\_info\* info)**

This API returns the metadata info related to a specific NKV instance. Nkv\_handle and instance\_uuid combination should be enough to detect a specific NKV instance. Mainly needed to know if NKV instance is still running for example to cleanup any object level lock.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **instance\_uuid --** Unique NKV instance identifier

IN/OUT **info** - nkv\_instance\_info structure containing meta information for a NKV instance, caller

is responsible to allocate this.

struct **nkv\_instance\_info** {

//Instance hosting node name or ip

char\* host\_name\_ip;

//Instance end-point

uint32\_t host\_port;

// nkv instance uuid

uint64\_t instance\_uuid;

//Instance creation time

time\_t mtime;

//NKV instance will be periodically updating this heart-beat time to let other instance know it is alive

// this is mainly needed to override a lock if owner instance is crashed before unlocking. Following

// attribute tells the duration in seconds last heart-beat updated

uint64\_t last\_hb\_duration;

};

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_INSTANCE\_NOT\_FOUND – NKV instance doesn’t exist

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_list\_instance\_info**

**int32\_t nkv\_list\_instance\_info (uint64\_t nkv\_handle, const char\* host\_name\_ip , uint32\_t index, nkv\_instance\_info\* info, uint32\_t\* num\_instances)**

This API returns the metadata info related to all NKV instances running/registered against a nkv handle. Mainly needed to know how many NKV/application instances are running/registered and where.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **host\_name\_id –** Filtered by Host name or IP, could be NULL

IN **index --** Start offset index in the list, first call it will be 0 and then added with

num\_instances returned.

IN/OUT **info** – Array of nkv\_instance\_info structure containing meta information for a NKV instance.

Caller is responsible to allocate the array and pass the number of allocated elements in

num\_instances field.

struct **nkv\_instance\_info** {

//Instance hosting node name or ip

char\* host\_name\_ip;

// nkv instance uuid

uint64\_t instance\_uuid;

//Instance creation time

time\_t mtime;

//NKV instance will be periodically updating this heart-beat time to let other instance know it is alive

// this is mainly needed to override a lock if owner instance is crashed before unlocking. Following

// attribute tells the duration in seconds last heart-beat updated

uint64\_t last\_hb\_duration;

};

IN/OUT **num\_instances –** Total number of pre-allocated instances as IN param and actual

number of element as OUT

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_INSTANCE\_NOT\_FOUND – NKV instance doesn’t exist

NKV\_ERR\_BUFFER\_SMALL – Buffer is not sufficient to accommodate all profiles

NKV\_INVALID\_INDEX – Start index is invalid

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_close**

**int32\_t nkv\_close (uint64\_t nkv\_handle, uint64\_t instance\_uuid)**

This API destroys the NKV client side data structures related to the nkv\_handle supplied. This also de-register instance and node from the NKV metadata structure.

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **instance\_uuid --** Unique NKV instance identifier

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_NOT\_OPENED – NKV instance is not opened

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_get\_version\_info**

**int32\_t nkv\_get\_version\_info (uint64\_t nkv\_handle, uint64\_t instance\_uuid, uint32\_t\* major\_v, uint32\_t\* minor\_v)**

This API tells about the major and minor version this app/instanceid combination is running with..

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **instance\_uuid --** Unique NKV instance identifier

OUT **major\_v** – major version of this NKV library

OUT **minor\_v** – minor version of this NKV library

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_NULL\_INPUT – major\_v and/or minor\_v is NULL

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_get\_supported\_feature\_list**

**int32\_t nkv\_get\_supported\_feature\_list (uint64\_t nkv\_handle, nkv\_feature\_list\* features)**

This API tells the supported feature list of NKV.

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

OUT **features** - nkv\_feature\_list structure containing supported features of NKV

struct **nkv\_feature\_list** {

uint32\_t data\_placement:1;

uint32\_t erasure\_coding:1;

uint32\_t replication:1;

uint32\_t recovery:1;

uint32\_t rebalance:1;

uint32\_t nic\_load\_balance:1;

uint32\_t nic\_failover:1;

uint32\_t authentication:1;

uint32\_t qos:1;

uint32\_t indexing:1;

uint32\_t iops\_optimized:1;

uint32\_t bw\_optimized:1; //default

uint32\_t lat\_optimized:1;

};

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_set\_required\_feature\_list**

**int32\_t nkv\_set\_required\_feature\_list (uint64\_t nkv\_handle, nkv\_feature\_list\* features)**

Application will use this API to set the feature list it will be using out of the supported feature list published by NKV. This way NKV will know for example application will be running on pass-through or non-pass-through mode.

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **features** - nkv\_feature\_list structure containing the required features of NKV needed by app

struct **nkv\_feature\_list** {

uint32\_t data\_placement:1;

uint32\_t erasure\_coding:1;

uint32\_t replication:1;

uint32\_t recovery:1;

uint32\_t rebalance:1;

uint32\_t nic\_load\_balance:1;

uint32\_t nic\_failover:1;

uint32\_t authentication:1;

uint32\_t qos:1;

uint32\_t indexing:1;

uint32\_t iops\_optimized:1;

uint32\_t bw\_optimized:1; //default

uint32\_t lat\_optimized:1;

};

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_FEATURE\_NOT\_SUPPORTED – Feature requested not supported by NKV

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_get\_supported\_key\_length\_info**

**int32\_t nkv\_get\_supported\_key\_length\_info (uint64\_t nkv\_handle, uint64\_t\* min, uint64\_t\* max, uint64\_t\* optimal)**

This API tells minimum, maximum and optimal key length supported by the nkv device related to the nkv\_handle supplied.

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

OUT **min** – Minimum key length supported

OUT **max** – Maximum key length supported

OUT **optimal** – Optimal key length, beyond this length performance will be impacted.

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_get\_supported\_value\_length\_info**

**int32\_t nkv\_get\_supported\_value\_length\_info (uint64\_t nkv\_handle, uint64\_t\* min, uint64\_t\* max, uint64\_t\* optimal\_min, uint64\_t\* optimal\_max)**

This API tells minimum, maximum and optimal range of value length supported by the nkv device related to the nkv\_handle supplied.

**REQUIRED**

Yes.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

OUT **min** – Minimum value length supported

OUT **max** – Maximum value length supported

OUT **optimal\_min** – Optimal minimum value length, below this length performance will be

impacted. The optimal size is calculated based on the iops/bw/latency

requirement set by the application previously.

OUT **optimal\_max** – Optimal maximum value length, beyond this length performance will be

The optimal size is calculated based on the iops/bw/latency

requirement set by the application previously.

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_physical\_container\_list**

**int32\_t nkv\_physical\_container\_list (uint64\_t nkv\_handle, uint32\_t index, nkv\_container\_info \*cntlist, uint32\_t \*cnt\_count )**

This API gives back the underlying remote physical container list for NKV. Application wants to use NKV in pass-through mode needs to use this api to get a list of physical containers for data placement.

**REQUIRED**

Yes, for **pass-through** mode.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **index** – Start index of the physical container list

IN/OUT **cntlist** – Array of physical container information. Needs to be pre-allocated

by caller for ‘cnt\_count’ number of entries.

struct **nkv\_container\_transport** {

//Container transport path id , generated by NKV

int32\_t network\_path\_id;

//Container transport family, 2 = ipv4 or 10 =ipv6

int8\_t addr\_family;

//Container transport speed, 0 = 1Gb, 1 = 10Gb, 2 = 50Gb, 3 = 100Gb

int8\_t speed;

//Container transport status, 0 = Down , 1 = Up

int8\_t status;

//Container transport aligned to numanode#

int8\_t numa\_node;

}

struct **nkv\_container\_info** {

// UUID of the container

uint64\_t container\_id;

// Name of the container

char\* container\_name;

// Physical target node hosting the container

char\* nkv\_target\_name;

// UUID of the physical node hosting the container

uint64\_t nkv\_target\_id;

// Container status, OK, DOWN

uint8\_t container\_status;

//Percentage space available on the container

uint8\_t container\_space\_available\_percentage;

// IN/OUT, number of nkv\_container\_transport\* allocated

int8\_t num\_container\_transport;

// Container transport details

nkv\_container\_transport\* transport\_list;

}

**IN/OUT** **cnt\_count** – Number of entries pre-allocated by caller and NKV will populate back

the actual number of physical container entries present in the cntlist buffer

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_NO\_ENTRY\_FOUND – Physical container doesn’t exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_BUFFER\_SMALL – Buffer is not sufficient to accommodate all profiles

NKV\_INVALID\_INDEX – Start index is invalid

NKV\_ERR\_NULL\_INPUT – cnt\_count is NULL

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_register\_for\_event**

**int32\_t nkv\_register\_for\_event (uint64\_t nkv\_handle, uint64\_t instance\_uuid, kv\_event\_function\* event\_func)**

This API registers an application callback function for an application instance. This function will be called back by NKV in case of any event (like configuration change etc.) generated by NKV fabric manager.

**REQUIRED**

Optional, for **pass-through** mode.

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **instance\_uuid --** Unique NKV instance identifier

IN/OUT **event\_func** – Application supplied kv\_event\_function\*

|  |  |
| --- | --- |
| typedef struct {  // Event code, 1 = config change, …. | |
| Int32\_t event\_code; |  |
| //Event data if any, NULL otherwise  void\* event\_data; |  |
|  |  |
| // private data address if any passed by caller |  |
| void \*private\_data; |  |
|  |  |
| } **nkv\_event\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_event\_cb)(nkv\_event\_construct\* ev);

// Application private data that application wants to get back with nkv\_event\_cb

void \*private\_data;

} kv\_event\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_NO\_ENTRY\_FOUND – Physical container doesn’t exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_NULL\_INPUT – event\_function is NULL

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* 1. **NKV IO APIs**

Following are the APIs for passing IO over NKV.

* + 1. **nkv\_retrieve\_kvp**

**int32\_t nkv\_retrieve\_kvp (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, const nkv\_retrieve\_option\* opt, nkv\_value\* value)**

This API retrieves a key value pair value of a given key.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

struct **nkv\_io\_context** {

//Using NKV in pass-through mode ? 0 – non-pass-through, 1 – pass-through

int8\_t is\_pass\_through;

// UUID of the physical container, needed for pass-through mode

uint64\_t container\_id;

// UUID of the physical node hosting the container, needed for pass-through mode

uint64\_t nkv\_target\_id;

//Container transport path id, not needed if nic\_load\_balance and nic\_failover feature is enabled

int32\_t network\_path\_id;

//NKV generated Key space Id, not needed for pass-through mode

int32\_t ks\_id;

}

IN **key** -- Key for which key value pair information needed.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN **opt** -- nkv\_retrieve\_option structure for specifying retrieve option.

struct **nkv\_retrieve\_option** {

int8\_t nkv\_retrieve\_decompress:1;

int8\_t nkv\_retrieve\_decrypt:1;

int8\_t nkv\_compare\_crc:1;

int8\_t nkv\_retrieve\_delete:1;

}

IN/OUT **value** – nkv\_value structure containing the value for the key. Needs to be

pre-allocated by user.

struct **nkv\_value** {

//The value buffer allocated by user

void \*value;

//The length of the value buffer in bytes

uint64\_t length;

//The actual length of the key value object in bytes

uint64\_t actual\_length;

}

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key, value, opt or ioctx is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key does not exist

NKV\_ERR\_OPTION\_COMPRESSION\_NOT\_SUPPORTED – Compression is not supported

NKV\_ERR\_OPTION\_ENCRYPTION\_NOT\_SUPPORTED – No encryption support

NKV\_ERR\_OPTION\_CRC\_NOT\_SUPPORTED – No crc support

NKV\_ERR\_OPTION\_DEL\_AFTER\_GET\_NOT\_SUPPORTED – Atomic delete after get is not supported

NKV\_ERR\_BUFFER\_SMALL – Value buffer is shorter than the actual object value length

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_retrieve\_kvp\_async**

**int32\_t nkv\_retrieve\_kvp\_async (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, const nkv\_retrieve\_option\* opt, nkv\_value\* value, nkv\_postprocess\_function\* post\_fn)**

This API retrieves a key value pair value of a given key asynchronously. The function will return immediately regardless of whether the pair is retrieved or not. The final execution results are returned to post process function through **nkv\_aio\_construct**.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information needed.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN **opt** -- nkv\_retrieve\_option structure for specifying retrieve option.

struct **nkv\_retrieve\_option** {

int8\_t nkv\_retrieve\_decompress:1;

int8\_t nkv\_retrieve\_decrypt:1;

int8\_t nkv\_compare\_crc:1;

int8\_t nkv\_retrieve\_delete:1;

}

IN/OUT **value** – nkv\_value structure containing the value for the key. Needs to be

pre-allocated by user should be deallocated from the callback nkv\_aio\_cb.

struct **nkv\_value** {

//The value buffer allocated by user

void \*value;

//The length of the value buffer in bytes

uint64\_t length;

//The actual length of the key value object in bytes

uint64\_t actual\_length;

}

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

|  |  |
| --- | --- |
| typedef struct { | |
| Int32\_t opcode; | // operation code, 0 = GET, 1 = PUT, 2= DEL |
| const kv\_key \* key; | // pointer for a key data structure |
| kv\_value \*value; | // pointer for a value data structure |
| Int32\_t result; | // return value (results) |
| void \*private\_data; | // private data address passed by caller |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} kv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key, value, opt, ioctx or post\_fn is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key does not exist

NKV\_ERR\_OPTION\_COMPRESSION\_NOT\_SUPPORTED – Compression is not supported

NKV\_ERR\_OPTION\_ENCRYPTION\_NOT\_SUPPORTED – No encryption support

NKV\_ERR\_OPTION\_CRC\_NOT\_SUPPORTED – No crc support

NKV\_ERR\_OPTION\_DEL\_AFTER\_GET\_NOT\_SUPPORTED – Atomic delete after get is not supported

NKV\_ERR\_BUFFER\_SMALL – Value buffer is shorter than the actual object value length

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_store\_kvp**

**int32\_t nkv\_store\_kvp (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, const nkv\_store\_option\* opt, nkv\_value\* value)**

This API stores a key value pair value of a given key.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be stored.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN **opt** -- nkv\_store\_option structure for specifying store option.

struct **nkv\_store\_option** {

int8\_t nkv\_store\_compressed:1;

int8\_t nkv\_store\_ecrypted:1;

int8\_t nkv\_store\_crc\_in\_meta:1;

int8\_t nkv\_store\_no\_overwrite:1;

int8\_t nkv\_store\_atomic:1

}

IN **value** – nkv\_value structure containing the value for the key. Needs to be

pre-allocated by user.

struct **nkv\_value** {

//The value buffer allocated by user

void \*value;

//The length of the value buffer in bytes

uint64\_t length;

//The actual length of the key value object in bytes (valid for retrieve case)

uint64\_t actual\_length;

}

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key, value, opt or ioctx is NULL

NKV\_ERR\_KVP\_EXIST – Key already exists and option says not to overwrite

NKV\_ERR\_OPTION\_COMPRESSION\_NOT\_SUPPORTED – Compression is not supported

NKV\_ERR\_OPTION\_ENCRYPTION\_NOT\_SUPPORTED – No encryption support

NKV\_ERR\_OPTION\_CRC\_NOT\_SUPPORTED – No crc support

NKV\_ERR\_SPACE – No space on device

NKV\_ERR\_VALUE – Given value is not supported

NKV\_ERR\_KVP\_LOCKED – Key value pair is locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_store\_kvp\_async**

**int32\_t nkv\_store\_kvp\_async (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, const nkv\_store\_option\* opt, nkv\_value\* value, nkv\_postprocess\_function\* post\_fn)**

This API stores a key value pair value of a given key asynchronously. The function will return immediately regardless of whether the pair is stored or not. The final execution results are returned to post process function through **nkv\_aio\_construct**.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be stored.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN **opt** -- nkv\_store\_option structure for specifying store option.

struct **nkv\_store\_option** {

int8\_t nkv\_store\_compressed:1;

int8\_t nkv\_store\_ecrypted:1;

int8\_t nkv\_store\_crc\_in\_meta:1;

int8\_t nkv\_store\_no\_overwrite:1;

int8\_t nkv\_store\_atomic:1

}

IN **value** – nkv\_value structure containing the value for the key. Needs to be

pre-allocated by user should be deallocated from the callback nkv\_aio\_cb.

struct **nkv\_value** {

//The value buffer allocated by user

void \*value;

//The length of the value buffer in bytes

uint64\_t length;

//The actual length of the key value object in bytes

uint64\_t actual\_length;

}

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

|  |  |
| --- | --- |
| typedef struct { | |
| Int32\_t opcode; | // operation code, 0 = GET, 1 = PUT, 2= DEL |
| const kv\_key \* key; | // pointer for a key data structure |
| kv\_value \*value; | // pointer for a value data structure |
| Int32\_t result; | // return value (results) |
| void \*private\_data; | // private data address passed by caller |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} kv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key, value, opt, ioctx or post\_fn is NULL

NKV\_ERR\_KVP\_EXIST – Key already exists and option says not to overwrite NKV\_ERR\_OPTION\_COMPRESSION\_NOT\_SUPPORTED – Compression is not supported

NKV\_ERR\_OPTION\_ENCRYPTION\_NOT\_SUPPORTED – No encryption support

NKV\_ERR\_OPTION\_CRC\_NOT\_SUPPORTED – No crc support

NKV\_ERR\_SPACE – No space on device

NKV\_ERR\_VALUE – Given value is not supported

NKV\_ERR\_KVP\_LOCKED – Key value pair is locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_delete\_kvp**

**int32\_t nkv\_delete\_kvp (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key)**

This API deletes a key value pair value of a given key.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be deleted.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key or ioctx is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key value pair doesn’t exist

NKV\_ERR\_KVP\_LOCKED – Key value pair is locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_delete\_kvp\_async**

**int32\_t nkv\_delete\_kvp\_async (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, nkv\_postprocess\_function\* post\_fn)**

This API deletes a key value pair value of a given key asynchronously. The function will return immediately regardless of whether the pair is stored or not. The final execution results are returned to post process function through **nkv\_aio\_construct**.

**REQUIRED**

Yes

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be stored.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

|  |  |
| --- | --- |
| typedef struct { | |
| Int32\_t opcode; | // operation code, 0 = GET, 1 = PUT, 2= DEL |
| const kv\_key \* key; | // pointer for a key data structure |
| kv\_value \*value; | // pointer for a value data structure |
| Int32\_t result; | // return value (results) |
| void \*private\_data; | // private data address passed by caller |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} nkv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key or ioctx is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key value pair doesn’t exist

NKV\_ERR\_KVP\_LOCKED – Key value pair is locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_lock\_kvp**

**int32\_t nkv\_lock\_kvp (uint64\_t nkv\_handle, uint64\_t instance\_uuid, nkv\_io\_context\* ioctx, const nkv\_key\* key, int32\_t forced\_lock, uint64\_t\* locked\_by, uin64\_t\* locked\_duration)**

This API locks a key value pair value of a given key.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **instance\_uuid --** Unique NKV instance identifier requesting to lock

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be locked.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

IN **forced\_lock –** override the lock if already locked

OUT **locked\_by** - The NKV instance uuid, in case the object is already locked

OUT **locked\_duration** - Number of seconds the object is been locked

Based on last two OUT params and heart-beat duration (in nkv\_instance\_info) application needs to determine if the lock is stale and call the api with forced\_lock = 1 to override.

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key or ioctx is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key value pair doesn’t exist

NKV\_ERR\_KVP\_ALREADY\_LOCKED – Key value pair is already locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_unlock\_kvp**

**int32\_t nkv\_unlock\_kvp (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key)**

This API unlocks a key value pair value of a given key.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key** -- Key for which key value pair information will be unlocked.

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key or ioctx is NULL

NKV\_ERR\_KVP\_NOT\_EXIST – Key value pair doesn’t exist

NKV\_ERR\_KVP\_NOT\_LOCKED – Key value pair is already locked

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_batch\_op\_async**

**int32\_t nkv\_batch\_op\_async (uint64\_t nkv\_handle, nkv\_io\_context\* ioctx, const nkv\_key\* key, nkv\_batch\_op\* ops, nkv\_postprocess\_function\* post\_fn)**

This API executes a batch of operation on a parent key asynchronously. For example, while storing a key we can use this api to store any meta and extended attributes along with the parent key. Operation on the list could be performed in order or out-of-order. In case of in-order batch op, we can execute a read-modify-write on a key all in one command and in the target side without sending any data back to client side. Batch operation on different non-related keys are not supported. The function will return immediately regardless of whether the batch op finished or not. The final execution results are returned to post process function through **nkv\_aio\_construct**.

**REQUIRED**

Optional, needed for performance improvement

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **ioctx** – nkv\_io\_context buffer required to perform IO on NKV.

IN **key --** Parent key for this batch operation

IN **ops --** nkv\_batch\_op structure for multiple key value operations.

struct **nkv\_op\_construct** {

// Type of operation, 0 = Retrieve, 1 = Store, 2 = Delete

uint32\_t op\_type;

// Input key

nkv\_key\* key;

// Input value, NULL for delete

nkv\_value\* value;

// Store related option, NULL for other operation

nkv\_store\_option\* store\_opt;

// Retrieve related option, NULL for other operation

nkv\_retrieve\_option\* store\_opt;

// Result after execution, Input should be 0;

int32\_t result;

};

struct **nkv\_batch\_op** {

// Total number of kv operation passed

uint32\_t number\_of\_ops;

// Execution of the ops will be in order it is provided or can be completed out-of-order.

// 0 = in-order, 1 = out-of-order

uint32\_t in\_order\_execution;

// individual op data structure

nkv\_op\_construct\* one\_op;

} ;

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

|  |  |
| --- | --- |
| typedef struct { | |
| Int32\_t opcode; | // operation code, 0 = GET, 1 = PUT, 2= DEL |
| const kv\_key \* key; | // pointer for a key data structure |
| kv\_value \*value; | // pointer for a value data structure |
| Int32\_t result; | // return value (results) |
| void \*private\_data; | // private data address passed by caller |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} nkv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_KS\_NOT\_EXIST – No key space with the supplied name exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_KS\_NOT\_OPEN – Key space not opened for this NKV instance

NKV\_ERR\_PHYSICAL\_CNT\_NOT\_EXIST – No physical container found in case of pass-through mode

NKV\_ERR\_PHYSICAL\_NODE\_NOT\_EXIST – No physical node found in case of pass-through mode

NKV\_ERR\_NETWORK\_PATH\_NOT\_EXIST – No network path found in case of pass-through mode

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported

NKV\_ERR\_NULL\_INPUT – key, ioctx, ops of post\_fn is NULL

NKV\_ERR\_KVP\_LOCKED – Key value pair is locked for any store operation

NKV\_ERR\_KVP\_BATCH\_OP\_FAILED – At least one operation failed within this batch

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_indexing\_add\_key\_async**

**int32\_t nkv\_indexing\_add\_key\_async (uint64\_t nkv\_handle, const nkv\_key\* key, const char\* bucket\_name, const char\* delimiter, nkv\_postprocess\_function\* post\_fn)**

This API adds a key to the NKV indexing structure for the bucket name supplied. NKV indexing logic will help the prefix based or hierarchical key listing to perform faster. Key will be added in lexicographically sorted way but asynchronously and that’s why immediate key listing may not be able to list the key.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **key** -- The actual Key to add in the indexing logic

IN **bucket\_name** – The parent bucket name for the key

IN **delimiter** – The delimiter char in the key for hierarchical listing. If not present

Supply “”

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

struct **nkv\_key** {

void \*key;

uint16\_t length;

}

|  |  |
| --- | --- |
| typedef struct {  // operation code, 0 = GET, 1 = PUT, 2= DEL, 3 = Index add, 4 = Index delete | |
| Int32\_t opcode;  // pointer for a key data structure |  |
| const kv\_key \* key;  // pointer for a value data structure |  |
| kv\_value \*value;  // return value (results) |  |
| Int32\_t result;  // private data address passed by caller |  |
| void \*private\_data; |  |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} nkv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported, probably unsupported characters

NKV\_ERR\_NULL\_INPUT – key or bucket name is NULL

NKV\_ERR\_BUCKET\_NOT\_EXIST – Bucket doesn’t exist

NKV\_ERR\_WRONG\_DELIMITER – Delimiter character not supported

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_indexing\_list\_keys**

**int32\_t nkv\_indexing\_list\_keys (uint64\_t nkv\_handle, const char\* bucket\_name, const char\* prefix, const char\* delimiter, const char\* start\_after, uint32\_t\* max\_keys, nkv\_key\* keys )**

This API list the keys for a bucket optionally based on prefix, delimiter and start\_after (lexicographically sorted) parameter supplied.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **bucket\_name** – The parent bucket name for the key

IN **prefix** – If it is not “”, all the keys prefix with this string will be listed

IN **delimiter** – The delimiter char in the key for hierarchical listing. If not present

supply “”

IN **start\_after** – List keys occur lexicographically after this key. In case of multiple

Iteration of providing the keys based on max\_keys, this field can be used

as index to start next iteration from.

IN/OUT **max\_keys --** maximum number of keys this api can pass at one shot. As out param

It will send out the number of keys it populated in the **‘keys’** array

IN/OUT **keys --** Application will allocate max\_keys number of empty nkv\_key structures.

Api will populate less than or equal to max\_keys number of keys.

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported, probably unsupported characters

NKV\_ERR\_NULL\_INPUT – keys structure or bucket name is NULL

NKV\_ERR\_BUCKET\_NOT\_EXIST – Bucket doesn’t exist

NKV\_ERR\_WRONG\_DELIMITER – Delimiter character not supported

NKV\_ERR\_PERMISSION -- a caller does not have required permission

* + 1. **nkv\_indexing\_delete\_key\_async**

**int32\_t nkv\_indexing\_delete\_key\_async (uint64\_t nkv\_handle, const nkv\_key\* key, const char\* bucket\_name, const char\* delimiter, nkv\_postprocess\_function\* post\_fn)**

This API deletes a key from the NKV indexing structure for the bucket name supplied. Key will be deleted asynchronously and that’s why immediate key listing may be still showing the key.

**REQUIRED**

Optional

**PARAMETERS**

IN **nkv\_handle** – A positive unique id for combination of nkv and the application.

IN **key** -- The actual Key to add in the indexing logic

IN **bucket\_name** – The parent bucket name for the key

IN **delimiter** – The delimiter char in the key for hierarchical listing. If not present

Supply “”

IN/OUT **post\_fn** – nkv\_postprocess\_function data structure is used to return the result. It

has user call back function returning the private\_data

|  |  |
| --- | --- |
| typedef struct {  // operation code, 0 = GET, 1 = PUT, 2= DEL, 3 = Index add, 4 = Index delete | |
| Int32\_t opcode;  // pointer for a key data structure |  |
| const kv\_key \* key;  // pointer for a value data structure |  |
| kv\_value \*value;  // return value (results) |  |
| Int32\_t result;  // private data address passed by caller |  |
| void \*private\_data; |  |
|  |  |
| } **nkv\_aio\_construct**; | |

typedef struct {

//nkv\_aio\_construct structure containing key and value pair along with operation return value

//num\_op represents number of nkv\_aio\_construct structure returned. Only in case of batch

// operation it could be greater than 1.

void (\*nkv\_aio\_cb)(nkv\_aio\_construct\* ops, int32\_t num\_op);

// Application private data that application wants to get back with nkv\_aio\_cb

void \*private\_data;

} nkv\_postprocess\_function;

**RETURNS**

NKV\_SUCCESS on success, following error codes on failure.

**ERROR CODE**

NKV\_ERR\_DEVICE\_NOT\_EXIST – No device with id nkv\_handle exists

NKV\_ERR\_INTERNAL -- Some internal error

NKV\_ERR\_IO – IO failure with NKV target node

NKV\_ERR\_KEY – Given key is not supported, probably unsupported characters

NKV\_ERR\_NULL\_INPUT – key or bucket name is NULL

NKV\_ERR\_BUCKET\_NOT\_EXIST – Bucket doesn’t exist

NKV\_ERR\_KEY\_NOT\_EXIST – key doesn’t exist

NKV\_ERR\_WRONG\_DELIMITER – Delimiter character not supported

NKV\_ERR\_PERMISSION -- a caller does not have required permission