SNIA®

Advancing storage &
information technology

Key Value Storage API Specification

Version 1.0

ABSTRACT: This SNIA document defines an application programing interface for Key Value Object drives.

This document has been released and approved by the SNIA. The SNIA believes that the ideas, methodologies and technologies described in this document accurately represent the SNIA goals and are appropriate for widespread distribution. Suggestions for revisions should be directed to http://www.snia.org/feedback/.

SNIA Technical Position

April 20, 2019

USAGE

DISCLAIMER

The information contained in this publication is subject to change without notice. The SNIA makes no warranty of any kind with regard to this specification, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The SNIA shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this specification.

Table of Contents

1 SCOPE 8

2 REFERENCES 9

3 DEFINITIONS, ABBREVIATIONS, AND CONVENTIONS 10

3.1 DEFINITIONS 10
3.1.1 Key Space 10
3.1.2 SSD 10
3.1.3 key value pair 10
3.2 KEYWORDS 10
3.2.1 mandatory 10
3.2.2 may 10
3.2.3 may not 10
3.2.4 need not 10
3.2.5 optional 10
3.2.6 shall 10
3.2.7 should 11
3.3 ABBREVIATIONS 11

4 OVERVIEW OF KVS API 12

4.1 OVERVIEW 12
4.2 KEY-VALUE ENTITIES 13
4.3 KEY SPACE 13
4.4 KEY GROUP 14
4.5 KEY VALUE PAIR 14

5 CONSTANTS & DATA STRUCTURES 15

5.1 TYPES 15
5.2 CONSTANTS 15
5.2.1 KVS_ALIGNMENT_UNIT 15
5.2.2 KVS_MAX_KEY_GROUP_BYTES 15
5.3 API RETURN VALUE (KVS_RESULT) 15
5.3.1 kvs_result 15
5.4 DATA STRUCTURES 16
5.4.1 kvs_api_version 16
5.4.2 kvs_context 17
5.4.3 kvs_key_order 17
5.4.4 kvs_option_key_space 18
5.4.5 kvs_option_delete 18
5.4.6 kvs_iterator_type 18
5.4.7 kvs_option_iterator 18
5.4.8 kvs_option_retrieve 19
5.4.9 kvs_store_type 19

5.4.10 kvs_association_type 20
5.4.11 kvs_assocititation 20
5.4.12 kvs_option_store 20
5.4.13 kvs_device_handle 20
5.4.14 kvs_key_space_handle 21
5.4.15 kvs_iterator_handle 21
5.4.16 kvs_key_space 21
5.4.17 kvs_key_space_name 21
5.4.18 kvs_device 22
5.4.19 kvs_exist_list 22
5.4.20 kvs_key_group_filter 22
5.4.21 kvs_iterator_list 23
5.4.22 kvs_key 24
5.4.23 kvs_postprocess_context 25
5.4.24 kvs_postprocess_function 25
5.4.25 kvs_value 25
5.4.26 kvs_kvp_info 26

6 KEY VALUE STORAGE APIS 27

6.1 OVERVIEW 27
6.2 DEVICE LEVEL APIS 28
6.2.1 kvs_open_device 28
6.2.2 kvs_get_device_info 29
6.2.3 kvs_close_device 30
6.2.4 kvs_get_device_capacity 31
6.2.5 kvs_get_device_utilization 32
6.2.6 kvs_get_min_key_length 33
6.2.7 kvs_get_max_key_length 34
6.2.8 kvs_get_min_value_length 35
6.2.9 kvs_get_max_value_length 36
6.2.10 kvs_get_optimal_value_length 37
6.2.11 kvs_create_key_space 38
6.2.12 kvs_delete_key_space 39
6.2.13 kvs_list_key_spaces 40
6.3 KEY SPACE-LEVEL APIS 41
6.3.1 kvs_open_key_space 41
6.3.2 kvs_close_key_space 42
6.3.3 kvs_get_key_space_info 43
6.3.4 kvs_get_kvp_info 44
6.3.5 kvs_retrieve_kvp 45
6.3.6 kvs_retrieve_kvp_async 46
6.3.7 kvs_store_kvp 47
6.3.8 kvs_store_kvp_async 48
6.3.9 kvs_delete_kvp 49
6.3.10 kvs_delete_kvp_async 50
6.3.11 kvs_delete_key_group 51
6.3.12 kvs_delete_key_group_async 52

6.3.13 kvs_exist_kv_pairs 53
6.3.14 kvs_exist_kv_pairs_async 54
6.4 ITERATOR FUNCTION CALLS 55
6.4.1 kvs_create_iterator 55
6.4.2 kvs_delete_iterator 57
6.4.3 kvs_iterate_next 58
6.4.4 kvs_iterate_next_async 59

TABLE OF FIGURES

Figure 1 Key-value Hierarchical Architecture 13
Figure 2 Key-value Entities 13
Figure 3 Fixed Key Length: kvs_iterator_key 24
Figure 4 Fixed Key Length: kvs_iterator_kvp 24
Figure 5 Variable Key Length: kvs_iterator_key 24
Figure 6 Variable Key Length: kvs_iterator_kvp 24

1 Scope

This specification defines the Application Programing Interface (API) for Key Value storage devices implementing the SNIA Object Drive protocol.

2 References

The following referenced documents are indispensable for the application of this document.

For references available from ANSI, contact ANSI Customer Service Department at (212) 642-49004980 (phone), (212) 302-1286 (fax) or via the World Wide Web at http://www.ansi.org.

NVMe
PCIe
SNIA IP Based Drive Management Specification

3 Definitions, abbreviations, and conventions

For the purposes of this document, the following definitions and abbreviations apply.

3.1 Definitions

3.1.1 Key Space

A collection of Key Value Pairs identified by a name and it is a unit of management in Key Value Storage see 4.3 (e.g., in NVMe a Namespace of type KeyValue)

3.1.2 SSD

Solid State Drive

3.1.3 key value pair

Object defined by a pair of key and value

3.2 Keywords

In the remainder of the specification, the following keywords are used to indicate text related to compliance:

3.2.1 mandatory

a keyword indicating an item that is required to conform to the behavior defined in this standard

3.2.2 may

a keyword that indicates flexibility of choice with no implied preference; “may” is equivalent to “may or may not”

3.2.3 may not

keywords that indicate flexibility of choice with no implied preference; “may not” is equivalent to “may or may not”

3.2.4 need not

keywords indicating a feature that is not required to be implemented; “need not” is equivalent to “is not required to”

3.2.5 optional

a keyword that describes features that are not required to be implemented by this standard; however, if any optional feature defined in this standard is implemented, then it shall be implemented as defined in this standard

3.2.6 shall

a keyword indicating a mandatory requirement; designers are required to implement all such mandatory requirements to ensure interoperability with other products that conform to this standard

3.2.7 should

a keyword indicating flexibility of choice with a strongly preferred alternative

3.3 Abbreviations

API	Application Programming Interface
KVS	Key Value Storage
NVMe	NVM Express (Non-Volatile Memory Express)
PCIe	PCI Express (Peripheral Component Interconnect Express)
SSD	Solid State Disk

4 Overview of KVS API

4.1 Overview

This document describes the Key Value Storage (KVS) Application Program Interface (API) specification for SSD storage devices with Object Drive based Key Value Storage. It provides a set of APIs that are portable across multiple vendor SSD products.

The KVS API provides management of the characteristics of the KVS instances to provide a common set of KVS instances. Once configured, all available KVS instances report the same characteristics.

Characteristics to provide to the host

1) Optimal STORE size (per key space)
2) Maximum number of keys/value size/key size/capacity (matrix) (aggregate – changes every time a Key Space is created/deleted)
3) Value granularity (per key space)
4) Minimum Key Length
5) Maximum Key Length
6) Minimum value Length
7) Maximum value Length
8) Total capacity (bytes) (aggregate and per key space)
9) Remaining capacity (bytes) (aggregate – changes every time a Key Space is created/deleted; and per key space)
10) Device Utilization

Characteristics of a device that is capable of Key Value storage are determined through a redfish implementation and allocation of a device to keyspaces is done through a KV management API. For an NVMe implementation there is at most one Keyspace per NVMe Namespace. For a SCSI implementation there is at most one Keyspace per SCSI LUN.

The library routines this document defines allow applications to create and use objects in SSDs while permitting portability. The library:

- Extends the C++ language with host and device APIs
- Provides support for Key Space, atomic operation, asynchronous operation, and callback

Library routines and environment variables provide the functionality to control the behavior of KVS. Figure 1 shows the hierarchical KVS architecture.

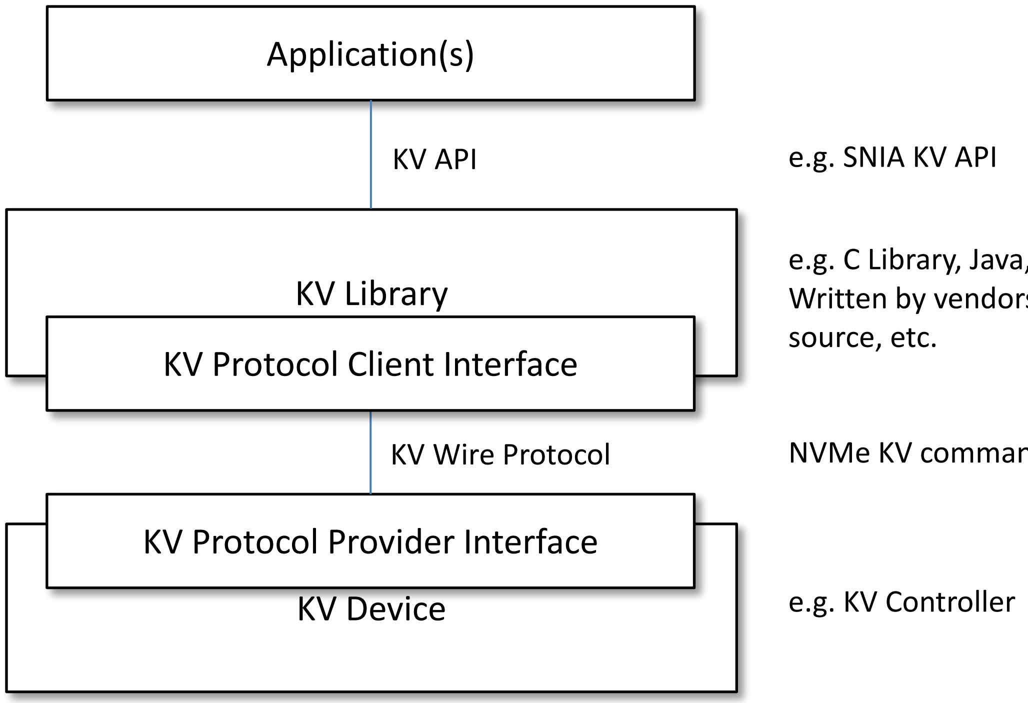
Application(s)
KV API
e.g. SNIA KV API
KV Library
e.g. C Library, Java, etc.
Written by vendors, open source, etc.
KV Protocol Client Interface
KV Wire Protocol
NVMe KV command set
KV Protocol Provider Interface
KV Device
e.g. KV Controller

Figure 1 Key-value Hierarchical Architecture

4.2 KEY-VALUE ENTITIES

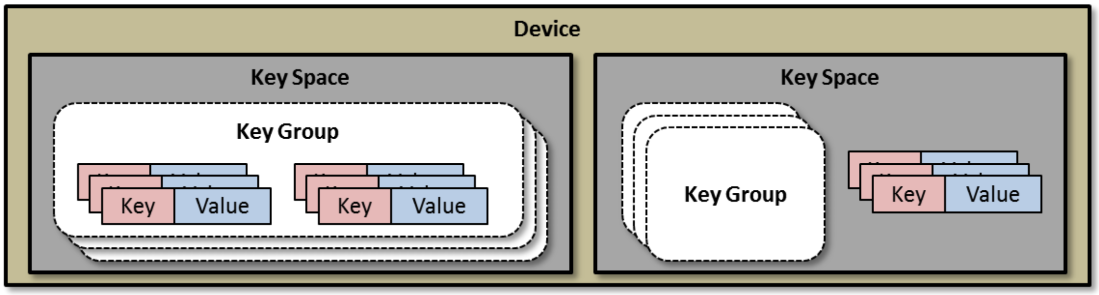

Figure 2 Key-value Entities

A Key-value device is a physical or logical storage device such as a HDD, SSD, or an NVM Set which has a native storage command protocol of a key-value interface. A Key Space is created from a portion or all of a Key Value device. Form factors (e.g., 2.25”, 2.5”, M.2, M.3, and HHHL) or command protocols (e.g., SATA, SCSI, NVMe, and NVMoF) are beyond the scope of this specification.

4.3 Key Space

A Key Space defines the uniqueness of keys (i.e., Keys shall be unique within a Key Space). A Key Space is associated with the specific configuration (e.g., key size, value size, capacity) with which it was created. Different Key Spaces in a device may be created with different configurations. A Key Space contains a collection of Key Value Entities (i.e., Key Value Pairs, or Key Groups) that are managed as a single entity (e.g., NVMe namespace, SCSI LUN, or disk partition). A device is able to simultaneously have multiple Key Spaces. A Key-value device shall support at least one Key Space. A Key Space is associated with a specified amount of capacity.

4.4 Key Group

A Key Group is a logical set of Key Value Pairs within a Key Space which applications are able to dynamically create. Key Groups are optional. This is able to be used to represent a shard, a document collection, an iterator, etc. A Key Group is specified by specific bits set to a given value in the key. The Key Group may be accessed using a call that specifies a mask of the bits in the key which defines the key group field, and a key group identifier identifying which Key Group is being accessed. A Key Space is able to simultaneously have multiple Key Groups. The Key Group field starts at the MSB and the size of the key group field is of byte granularity.

4.5 Key Value Pair

A Key Value Pair is an entity consisting of a key and a value. It is a unit of access. A key is application-defined and unique within a Key Space. The key length is able to be fixed or variable and its maximum is limited. A value length is variable and its maximum is limited.

5 Constants & Data Structures

This section defines Key-value SSD core constants, data structures, and functions.

5.1 Types

5.2 Constants

5.2.1 KVS_ALIGNMENT_UNIT

This is an alignment unit. An offset of *value* is required to be a multiple of this value.

5.2.2 KVS_MAX_KEY_GROUP_BYTES

The maximum number of bytes used for Key Group_bytes. This is set when a device is opened (e.g., if KVS_MAX_KEY_GROUP_BYTES is 3, any 3 bytes out of a key are able to be used to define a Key Group) and is the same for all Key Spaces in the device.

5.3 API return value (kvs_result)

5.3.1 kvs_result

An API returns a return value after finishing its operation. Two types of return value are returned. One is returned after the command is sent and the other after the command completes.

Return value details are discussed in each command section.

```
typedef enum {
KVS_SUCCESS                       0        // Successful
KVS_ERR_BUFFER_SMALL              0x001    // buffer space is not enough
KVS_ERR_DEV_CAPAPCITY             0x002    // device does not have enough space. Key Space size is too
                                           big
KVS_ERR_DEV_NOT_EXIST             0x003    // no device with the dev_hd exists
KVS_ERR_KS_CAPACITY               0x004    // key space does not have enough space
KVS_ERR_KS_EXIST                  0x005    // key space is already created with the same name
KVS_ERR_KS_INDEX                  0x006    // index is not valid
KVS_ERR_KS_NAME                   0x007    // key space name is not valid
KVS_ERR_KS_NOT_EXIST              0x008    // key space does not exist
KVS_ERR_KS_NOT_OPEN               0x009    // key space does not open
KVS_ERR_KS_OPEN                   0x00A    // key space is already opened
KVS_ERR_ITERATOR_FILTER_INVALID   0x00B    // iterator filter(match bitmask and pattern) is not valid
KVS_ERR_ITERATOR_MAX              0x00C    // the maximum number of iterators that a device supports
                                           is opened
KVS_ERR_ITERATOR_NOT_EXIST        0x00D    // the iterator Key Group does not exist
KVS_ERR_ITERATOR_OPEN             0x00E    // iterator is already opened
KVS_ERR_KEY_LENGTH_INVALID        0x00F    // key is not valid (e.g., key length is not supported)
KVS_ERR_KEY_NOT_EXIST             0x010    // key does not exist
KVS_ERR_OPTION_INVALID            0x011    // an option is not supported in this implementation
KVS_ERR_PARAM_INVALID             0x012    // null input parameter
KVS_ERR_SYS_IO                    0x013    // I/O error occurs
KVS_ERR_VALUE_LENGTH_INVALID      0x014    // value length is out of range
KVS_ERR_VALUE_OFFSET_INVALID      0x015    // value offset is out of range
KVS_ERR_VALUE_OFFSET_MISALIGN     0x016    //offset of value is required to be aligned to
ED                                         KVS_ALIGNMENT_UNIT
KVS_ERR_VALUE_UPDATE_NOT_ALL      0x017    // key exists but value update is not allowed
OWED
} kvs_result;
```

5.4 Data Structures

5.4.1 kvs_api_version

```
typedef struct {
  uint8_t  major;            // API library major version number
  uint8_t  minor;            // API library minor version number
  uint8_t micro;             // API library micro version number
  } kvs_api_version;
```

The *kvs_api_version* structure defines the API library version. For example the kvs_api_version for KV-API version 0.17 would be 0x001100.

5.4.2 kvs_context

```
typedef enum {
  KVS_CMD_DELETE               =0x01,
  KVS_CMD_DELETE_GROUP         =0x02,
  KVS_CMD_EXIST                =0x03,
  KVS_CMD_ITER_CREATE          =0x04,
  KVS_CMD_ITER_DELETE          =0x05,
  KVS_CMD_ITER_NEXT            =0x06,
  KVS_CMD_RETRIEVE             =0x07,
  KVS_CMD_STORE                =0x08,
} kvs_context;
```

kvs_context sets up opcode in API level for key value operation.

5.4.3 kvs_key_order

```
typedef enum {
  KVS_KEY_ORDER_NONE        =0,  // [DEFAULT] key ordering is not defined in
                                 a Key Space
  KVS_KEY_ORDER_ASCEND,     =1,
                                 // kvp are sorted in ascending key order in
  KVS_KEY_ORDER_DESCEND     =2,  a Key Space
} kvs_key_order;                 // kvp are sorted in descending key order in
                                 a Key Space
```

This enumeration specifies the ordering of keys returned .

- KVS_KEY_ORDER_NONE, no key order is defined in a key space.
- KVS_KEY_ORDER_ASCEND, key value pairs are sorted in ascending key order in a Key Space
- KVS_KEY_ORDER_DESCEND, key value pairs are sorted in descending key order in a Key Space

5.4.4 kvs_option_key_space

```
typedef struct {
   kvs_key_order   ordering;                    // key ordering option in Key Space

} kvs_option_key_space;
```

A user is able to define the ordering of keys returned.

5.4.5 kvs_option_delete

```
typedef struct {
      bool      kvs_delete_error;                  //[OPTION] return error when the key
                                                    does not exist
  } kvs_option_delete;
```

The application is able to specify a delete operation option.

- kvs_delete_error set to TRUE specifies that an operation deletes the key-value pair or if the key does not exist, the device return KVS_ERR_KEY_NOT_EXIST error code. kvs_delete_error set to FALSE specifies that an operation deletes the key if it exists and always returns success even if the key does not exist.

5.4.6 kvs_iterator_type

```
typedef enum {
   KVS_ITERATOR_KEY          =0,    // [DEFAULT] iterator command retrieves only
                                     key entries without values
KVS_ITERAOR_KEY_VALUE   =1,    // iterator command retrieves key and value
                                     pairs
} kvs_iterator_type;
```

5.4.7 kvs_option_iterator

```
typedef struct {
  kvs_iterator_type iter_type;          // iterator type

} kvs_option_iterator;
```

5.4.8 kvs_option_retrieve

```
typedef struct {

    bool      kvs_retrieve_delete;              // [OPTION] retrieve the value of the
                                                key value pair and delete the key
                                                value pair
  } kvs_option_retrieve;
```

The application is able to specify a retrieve operation option.

- *kvs_retrieve_delete* set to TRUE specifies that an operation retreives the key-value pair and the key value pair is atomically deleted after completing the retreive. *kvs_retrieve_delete* set to FALSE specifies that an operation retreives the key-value pair and no deletion is atomically performed.

5.4.9 kvs_store_type

```
typedef enum {
  KVS_STORE_POST              =0,    // [DEFAULT]
  KVS_STORE_UPDATE_ONLY       =1,
  KVS_STORE_NOOVERWRITE       =2,
  KVS_STORE_APPEND            =3,
} kvs_store_type;;
```

The application is able to specify a store operation option.

- KVS_STORE_POST: if the key exist, the operation overwrites the value. if the key does not exist, it creates the key value pair.
- KVS_STORE_UPDATE_ONLY: If the key exist, the operation overwrites the value. If the key does not exist, it returns KVS_KEY_NOT_EXIST error.
- KVS_STORE_NOOVERWIRTE: if the key exist, the operation returns KVS_ERR_VALUE_UPDATE_NOT_ALLOWED. If the key does not exist, it creates the key value pair.
- KVS_STORE_APPEND: if the key exist, the operation appends the value to the existing value. if the key does not exist, it creates the key value pair.

5.4.10 kvs_association_type

```
typedef enum {
   KVS_NOASSOCIATION           =0,     // no association
KVS_ASSOCIATION_STREAM     =1,     // stream association
} kvs_association_type;;
```

The application is able to specify an association option.

- KVS_NOASSOCIATION: no association defined
- KVS_ASSOCIATION_STREAM: key value pair associated with stream

5.4.11 kvs_associtation

```
typedef struct {
     kvs_association_type  assoc_type;        // association type for a group of
                                              associated key value pairs.
    uint16_t                    assoc_hint;        // association hint(e.g., stream id)
} kvs_association;
```

The application is able to specify an association type and hint.

5.4.12 kvs_option_store

```
typedef struct {
   kvs_store_type        st_type;        // store operation type (refer to 5.4.10)

  kvs_association        *assoc;         // association (refer to 5.4.12)
} kvs_option_store ;
```

The application is able to define store operation options.

5.4.13 kvs_device_handle

```
typedef void* kvs_device_handle;                   // type definition of kvs_device_handle
```

A *kvs_device_handle* is a vendor-specific opaque data structure pointer. API programmers may define a private vendor-specific data structure, which may contain the device id and other device-related information, and use this pointer type as a device handle.

5.4.14 kvs_key_space_handle

typedef void **kvs_key_space_handle**;*	*// type definition of kvs_key_space_handle*

A *kvs_key_space_handle* is a vendor-specific opaque data structure pointer. API programmers may define a private vendor-specific data structure, which may contain the key space id and other key space related information, and use this pointer type as a key space handle.

5.4.15 kvs_iterator_handle

typedef void **kvs_iterator_handle**;*	*// type definition of kvs_iterator_handle*

A *kvs_iterator_handle* is a vendor-specific opaque data structure pointer. API programmers may define a private vendor-specific data structure, which contains the iterator id and other iterator related information, and use this pointer type as an iterator handle.

5.4.16 kvs_key_space

typedef struct {	
bool_t opened;	*// is this Key Space opened*
uint64_t capacity;	*// Key Space capacity in bytes*
uint64_t free_size;	*// available space of Key Space in bytes*
uint64_t count;	*// # of Key Value Pairs that exist in this Key Space*
*kvs_key_space_name *name;*	*// Key Space name*
*} **kvs_key_space;***	

A Key Space is a unit of management and represents a collection of Key Value Pairs or Key Groups.

5.4.17 kvs_key_space_name

typedef struct {	
uint32_t name_len;	*// Key Space name length*
*kvs_key_space_name*name;*	*//Key Space name specified by the application*
*} **kvs_key_space_name;***	

This structure contains Key Space name information for return value of kvs_list_key_space() API. The name is of length name_len and if it is null terminated the null is part of the length. A device is not required to check the uniqueness of Key Space name.

5.4.18 kvs_device

```
typedef struct {
   Uint64_t   capacity;             // device capacity in bytes
   Uint64_t   unalloc_capacity;     // device capacity in bytes that has not been

                                    allocated to any key space

   uint32_t  max_value_len;         // max length of value in bytes that device is able to
                                    support
   uint32_t   max_key_len;          // max length of key in bytes that device is able to
                                    support
   uint32_t  optimal_value_len;     // optimal value size
   uint32_t  optimal_value_         // optimal value granularity
granularity;
   void      *extended_info;        // vendor specific extended device information.
   } kvs_device;
```

kvs_device structure represents a device and has device-wide information.

5.4.19 kvs_exist_list

```
typedef struct {
uint32_t      num_keys;      // the number of key entries in the list
kvs_keys *keys;              // keys checked for existence
uint32_t      length;        // input buffer size(result_buffer) and returned buffer size
 uint8_t *result_buffer;     // exist status info
} kvs_exist_list;
```

A *kvs_exist_list* structure is used to check whether keys exist in the KV device. The *result_buffer* field presents the existence of the keys. Each bit in the result buffer is set to one if the key exists and set to zero if the key does not exist.

5.4.20 kvs_key_group_filter

```
typedef struct {

   uint8                                            // bit mask for bit pattern to use
bitmask[KVS_MAX_KEY_GROUP_BYTES];
   uint8                                            // bit pattern for filter
bit_pattern[KVS_MAX_KEY_GROUP_BYTES];
   } kvs_key_group_filter;
```

This structure defines Key Group information for *kvs_create_iterator() that* sets up a Key Group of keys matched with a given *bit_pattern* within a range of bits defined by the *bitmask and for kvs_delete_key_group() such that it is able to delete a group of key-value pairs*. Bitmask is to be set in multiple of 8 bits starting from the MSB of the 32 bit value. For more details, see *kvs_create_iterator()* (section 6.4.1) and kvs_delete_key_group() (section 6.3.10).

5.4.21 kvs_iterator_list

typedef struct {	
uint32_t num_entries;	*// the number of iterator entries in the list*
bool_t end;	*// represent if there are more keys to iterate (end =0) or not (end = 1)*
uint32_t size;	*// the it_list buffer size as an input and returned data size in the buffer in bytes*
*uint8_t *it_list;*	*// iterator list.*
} ***kvs_iterator_list****;*	

kvs_iterator_list represents entries within an iterator Key Group. It is used for retrieved iterator entries as a return value for *kvs_interator_next()* operation. *num_entries* specifies how many entries in the returned iterator list(*it_list*). *size* specifies buffer size of *it_list* as an input and specifies the total amount of data that is returned in bytes as an output. *end* indicates that no more iterator items exist. When *end* is zero, host would re-run *kvs_iterator_next()* to retrieve more data. *it_list* has *num_entries* of iterator elements as follows;

- *When key length is fixed, num_entries entries of <key>* when iterator is set with *KVS_ITERATOR_KEY* (Figure 3) and *num_entries* entries of *<key, value_length, value>* when iterator is set with *KVS_ITERATOR_KEY_VALUE* (Figure 4*)*
- *When keys have variable length, num_entries entries of <key_length, key>* when iterator is set with *KVS_ITERATOR_KEY* (Figure 5) and *num_entries* entries of *<key_length, key, value_length, value>* when iterator is set with *KVS_ITERATOR_KEY_VALUE* (Figure 6).

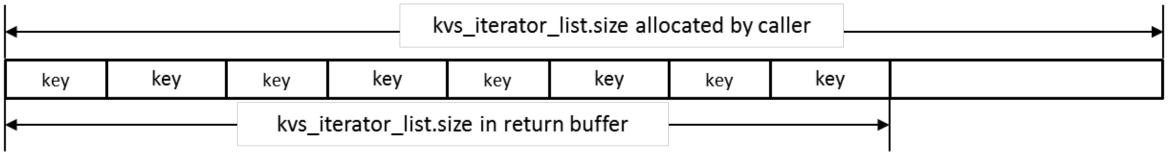
kvs_iterator_list.size allocated by caller
key
key
key
key
key
key
key
key
kvs_iterator_list.size in return buffer

Figure 3 Fixed Key Length: kvs_iterator_key

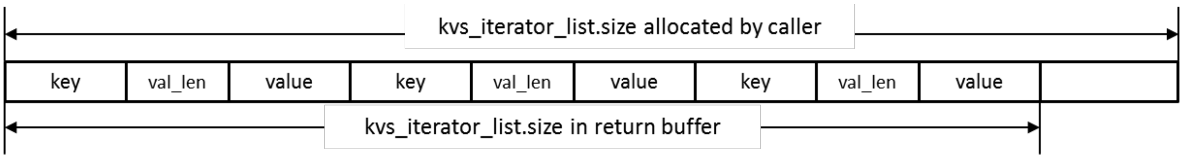
kvs_iterator_list.size allocated by caller
key
val_len
value
key
val_len
value
key
val_len
value
kvs_iterator_list.size in return buffer

Figure 4 Fixed Key Length: kvs_iterator_kvp

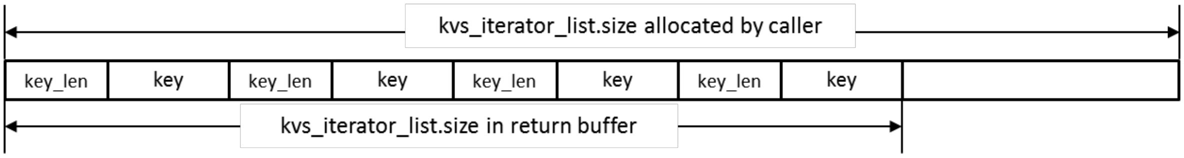
kvs_iterator_list.size allocated by caller
key_len
key
key_len
key
key_len
key
key_len
key
kvs_iterator_list.size in return buffer

Figure 5 Variable Key Length: kvs_iterator_key

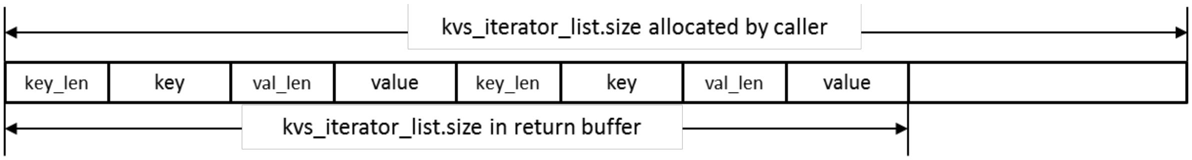
kvs_iterator_list.size allocated by caller
key_len
key
val_len
value
key_len
key
val_len
value
kvs_iterator_list.size in return buffer

Figure 6 Variable Key Length: kvs_iterator_kvp

5.4.22 kvs_key

```
typedef struct {

 void *key;                    // a void pointer refers to a key byte string
   uint16_t length;            // key length in bytes

   } kvs_key;
```

A key consists of a void pointer and its length. For a Key Space with variable keys (i.e., character string or byte string), the void *key* pointer holds a byte string without a null termination, and the integer variable of *length* holds the string byte count. The void *key* pointer is required not to be a null pointer.

5.4.23 kvs_postprocess_context

```
typedef struct {
   kvs_context context;                    // operation type
   kvs_key_space_handle *ks_hd;            // key space handle
   kvs_key *key;                           // key data structure
   kvs_value *value;                       // value data structure
   void *option;                           // operation option
   void *private1;                         // a pointer passed from a user
   void *private2                          // a pointer passed from a user
   kvs_result result;                      // IO result
   kvs_iterator_handle*iter_hd;            // iterator handle
} kvs_postprocess_context;
```

kvs_postprocess_context is IO context that carries IO information including key and value pairs and operation return value. It is mainly used for post process function.

> *Note: Async is for performance benefit. Multi-thread may cover it but we could reduce system resource utilizations with higher performance. Also more scalable. E.g. SPDK.*

5.4.24 kvs_postprocess_function

*typedef void(*kvs_postprocess_function)(kvs_postprocess_context *ctx)*	*// asynchronous notification callback (valid only for async I/O)*

kvs_postprocess_function is able to be called and specifies the tasks needing execution once an IO operation completes. Typical post-processing tasks send a signal to a thread to wake it up to implement synchronous IO semantics and/or call an application-defined notification function to implement asynchronous IO semantics.

5.4.25 kvs_value

typdef struct {	
*void *value;*	*// start address of buffer for value byte stream*
uint32_t length;	*// the length of buffer in bytes for value byte stream*
uint32_t actual_value_size;	*// actual value size in bytes that is stored in a device*
uint32_t offset;	*// [OPTION] offset to indicate the offset of value stored in device*
} ***kvs_value;***	

A value consists of a void pointer and a length. The value pointer refers to a byte string without null termination, and the length variable holds the byte count. The value pointer variable shall not be a null pointer. Offset specifies the offset within a value stored in the device. The offset is required to be aligned to KVS_ALIGNMENT_UNIT. If not, a KVS_ERR_VALUE_OFFSET_MISALIGNED error is returned.

5.4.26 kvs_kvp_info

```
typedef struct {
   uint16_t   key_len;                  // key length in bytes
   uint8_t    *key;                     // key
   uint32_t    value_len;               // value length in bytes
   } kvs_kvp_info;
```

This data structure contains key value pair properties associated with a key.

6 Key Value Storage APIs

6.1 Overview

This clause defines the core data structures for key-value device. A Key Space may be allocated from a single storage device, a storage array, an entry point into a cloud storage device or any other device that implements the KVS API. A Key Space is created using the kvs_create_keyspace API call. The Key Space is then opened using the kvs_open_keyspace API call.

6.2 Device level APIs

6.2.1 kvs_open_device

kvs_result kvs_open_device (char *URI, kvs_device_handle *dev_hd)

This API opens a KVS device. This API internally checks device availability and initializes it. It returns zero if successful. Otherwise, it returns an error code.

PARAMETERS

IN	URI	Universal Resource Identifier of a device
OUT	dev_hd	device handle

RETURNS

KVS_SUCCESS to indicate that device open is successful or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	the device does not exist
KVS_ERR_SYS_IO	communication with device failed
KVS_ERR_PARAM_INVALID	URI is NULL

6.2.2 kvs_get_device_info

kvs_result kvs_get_device_info(kvs_device_handle dev_hd, kvs_device *dev_info)

This function call retrieves the device information (e.g., kvs_device data structure).

PARAMETERS

IN	dev_hd	device handle
OUT	dev_info	kvs_device data structure (device information)

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.3 *kvs_close_device*

kvs_result kvs_close_device (kvs_device_handle dev_hd)

This API closes a KVS device. *dev_hd* is associated with an open device.

PARAMETERS

IN dev_hd	device handle

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device with the *dev_hd* exists
KVS_ERR_SYS_IO	communication with device failed

6.2.4 kvs_get_device_capacity

kvs_result kvs_get_device_capacity(kvs_device_handle dev_hd, uint64_t *dev_capacity)

This function call returns device capacity in bytes referenced by the given device handle.

PARAMETERS

IN	dev_hd	device handle
OUT	dev_capacity	device capacity

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.5 kvs_get_device_utilization

kvs_result kvs_get_device_utilization (kvs_device_handle dev_hd, uint32_t *dev_utilization)

This function call returns the device utilization (i.e, used ratio of the device) by the given device handle. The utilization is from 0(0.00% utilized) to 10000(100%).

PARAMETERS

IN	dev_hd	device handle
OUT	dev_utilization	device utilization

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.6 *kvs_get_min_key_length*

kvs_result kvs_get_min_key_length (kvs_device_handle dev_hd, uint32_t *min_key_length)

This function call returns the minimum length of key that the device supports.

PARAMETERS

IN	dev_hd	device handle
OUT	min_key_length	minimum key length that the device supports

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.7 *kvs_get_max_key_length*

kvs_result kvs_get_max_key_length (kvs_device_handle dev_hd, uint32_t *max_key_length)

This function call returns the maximum length of key that the device supports.

PARAMETERS

IN	dev_hd	device handle
OUT	max_key_length	maximum key length that the device support

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.8 *kvs_get_min_value_length*

kvs_result kvs_get_min_value_length (kvs_device_handle dev_hd, uint32_t *min_value_length)

This function call returns the minimum length of value that the device supports.

PARAMETERS

IN	dev_hd	device handle
OUT	min_value_length	minimum value length that the device supports

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.9 kvs_get_max_value_length

kvs_result kvs_get_max_value_length (kvs_device_handle dev_hd, uint32_t *max_value_length)

This function call returns the maximum length of value that the device supports.

PARAMETERS

IN	dev_hd	device handle
OUT	max_value_length	maximum value length that the device supports

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.10 kvs_get_optimal_value_length

kvs_result kvs_get_optimal_value_length (kvs_device_handle dev_hd, uint32_t *opt_value_length)

This function call returns the optimal length of value that the device supports. The device will perform best when the value size is the same as the optimal value size.

PARAMETERS

IN	dev_hd	device handle
OUT	opt_value_length	optimal value length that the device supports

RETURNS

KVS_SUCCESS for successful completion or an error code for error

ERROR CODE

KVS_ERR_DEV_NOT_EXIST	no device exists for the device handle
KVS_ERR_SYS_IO	communication with device failed

6.2.11 kvs_create_key_space

kvs_result kvs_create_key_space (kvs_device_handle dev_hd, kvs_key_space_name *key_space_name, uint64_t size, kvs_option_key_space opt)

This API creates a new Key Space in a device. An application needs to specify a unique Key Space name, and its capacity. The capacity is defined in bytes. A 0 (numeric zero) capacity means no limitation where device capacity limits actual Key Space capacity. The device assigns a unique id while an application assigns a unique name.

PARAMETERS

IN dev_hd	device handle
IN key_space_name	name of Key Space
IN size	capacity of a Key Space with respect to key value pair size (key size + value size) in bytes
IN opt	Key Space option

RETURNS

KVS_SUCCESS if a Key Space is created successfully or an error code for error.

ERROR CODE

KVS_ERR_DEV_CAPACITY	the Key Space size is too big
KVS_ERR_KS_EXIST	Key Space with the same name already exists
KVS_ERR_KS_NAME	Key Space name does not meet the requirement (e.g., too long (see 5.2.2))
KVS_ERR_DEV_NOT_EXIST	no device with the *dev_hd* exists
KVS_ERR_SYS_IO	communication with device failed
KVS_ERR_PARAM_INVALID	*name* or opt is NULL
KVS_ERR_OPTION_INVALID	Key Space option is not supported

6.2.12 kvs_delete_key_space

kvs_result kvs_delete_key_space (kvs_device_handle dev_hd, kvs_key_space_name *key_space_name)

This API deletes a Key Space identified by the given Key Space name. It deletes all Key Value Pairs within the Key Space as well as the Key Space itself. As a side effect of the delete operation, the Key Space is closed for all applications as the Key Space is no longer present in the device. It is recommended that all applications accessing a Key Space close the Key Space prior to deleting the Key Space.

PARAMETERS

IN dev_hd	device handle
IN key_space_name	Key Space name

RETURNS

KVS_SUCCESS if   a Key Space is deleted successfully or an error code for error

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *key_space_name* does not exist
KVS_ERR_DEV_NOT_EXIST	no device with the *dev_hd* exists
KVS_ERR_SYS_IO	communication with device failed

6.2.13 kvs_list_key_spaces

kvs_result kvs_list_key_spaces (kvs_device_handle dev_hd, uint32_t index, uint32_t buffer_size, kvs_key_space_name *names, uint32_t *ks_cnt)

For a KVS device, this API returns the names of Key Spaces up to the number that fit in the buffer specified in *buffer_size*. A device may define a unique order of Key Space names and index is defined relative to that order. The value of index may change if a Key Space is created or deleted. The *index* specifies a start list entry offset, buffer_size specifies the size of the *kvs_key_space_name* array, and *names* is a buffer to store name information. The ks_cnt specifies the number of Key Space names to return.

PARAMETERS

IN dev_hd	device handle
IN index	start index of Key Space as an input
IN buffer_size	buffer size of Key Space names
OUT names	buffer to store Key Space names. This buffer is required to be preallocated before calling this routine.
OUT ks_cnt	the number of *names* stored in the buffer

RETURNS

KVS_SUCCESS if the operation is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	no Key Space exists
KVS_ERR_DEV_NOT_EXIST	no device with the *dev_hd* exists
KVS_ERR_SYS_IO	communication with device failed
KVS_ERR_KS_INDEX	*index* is not valid
KVS_ERR_PARAM_INVALID	*names* or *ks_cnt* is NULL

6.3 Key Space-level APIs

6.3.1 kvs_open_key_space

kvs_result kvs_open_key_space (kvs_device_handle dev_hd, char *name, kvs_key_space_handle *ks_hd)

This API opens a Key Space with a given name. This API communicates with a device to initialize the corresponding Key Space. The device is capable of recognizing and initializing the Key Space. If the Key Space is already open, this API returns KVS_ERR_KS_OPEN.

PARAMETERS

IN dev_hd	Device handle
IN name	Key Space name
OUT ks_hd	Key Space handle

RETURNS

KVS_SUCCESS to indicate that device open is successful or an error code for error

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with the given *name* does not exist,
KVS_ERR_DEV_NOT_EXIST	No device with *dev_hd* exists
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KS_OPEN	Key Space has been opened already

6.3.2 kvs_close_key_space

kvs_result kvs_close_key_space (kvs_key_space_handle ks_hd)

This API closes a Key Space with a given Key Space handle. This API communicates with the device to close the corresponding Key Space. This API may clean up any internal Key Space states in the device. If the given Key Space was not open, this returns a KVS_ERR_KS_NOT_OPEN error.

PARAMETERS

IN ks_hd	Key Space handle

RETURNS

KVS_SUCCESS to indicate that closing a Key Space is successful or an error code for an error

ERROR CODE

KVS_ERR_KS_NOT_OPEN	Key space is not open
KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_DEV_NOT_EXIST	No device with *dev_hd* exists
KVS_ERR_SYS_IO	Communication with device failed

6.3.3 kvs_get_key_space_info

kvs_result kvs_get_key_space_info (kvs_key_space_handle ks_hd, kvs_key_space *ks)

This API retrieves Key Space information.

PARAMETERS

IN ks_hd	Key Space handle
OUT ks	Key Space information

RETURNS

KVS_SUCCESS to indicate that getting Key Space info is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_PARAM_INVALID	ks is NULL

6.3.4 kvs_get_kvp_info

kvs_result kvs_get_kvp_info (kvs_key_space_handle ks_hd, kvs_key *key, kvs_kvp_info *info)

This API retrieves key value pair properties. Key value pair properties includes a key length, a key byte stream, and a value length. Please refer to section *5.4.22 kvs_kvp_info* for details. This API is intended to be used when a buffer length for a value is not known. The caller should create kvs_kvp_info object before calling this API.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key to find for key value properties
OUT info	Key value pair properties

RETURNS

KVS_SUCCESS to indicate that retrieving key value pair properties is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_PARAM_INVALID	*key* or *info* is NULL
KVS_ERR_KEY_NOT_EXIST	*key* does not exist

6.3.5 kvs_retrieve_kvp

kvs_result kvs_retrieve_kvp (kvs_key_space_handle ks_hd, kvs_key *key, kvs_option_retrieve *opt, kvs_value *value)

This API retrieves a key value pair value with the given key. The value parameter contains output buffer information for the value. As an input, value.value contains the buffer to store the key value pair value and value.length contains the buffer size. The key value pair value is copied to value.value buffer and value.length is set to the retrieved value size. If the offset of value is not zero, the value of key value pair is copied into the buffer, skipping the first offset bytes of the value of key value pair. The offset is required to align to KVS_ALIGNMENT_UNIT. If the offset is not aligned, a KVS_ERR_VALUE_OFFSET_MISALIGNED error is returned and no data is transferred. If an allocated value buffer is not big enough to hold the value, the device will set actual_value_size to the size of the value, return KVS_ERR_BUFFER_SMALL and data is returned to the buffer up to the size specified in value.length.

The retrieve option is defined in 5.4.8 kvs_option_retreive.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key of the key value pair to get value
IN opt	retrieval option. It may be NULL. In that case, the default retrieval option is used.
OUT value	value to receive the key value pair's value from device

RETURNS

KVS_SUCCESS to indicate that retreive is successful or an error code for error.

ERROR CODE

KVS_ERR_VALUE_OFFSET_MISALIGNED	*kvs_value.offset* is not aligned to KVS_ALIGNMENT_UNIT
KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_BUFFER_SMALL	Buffer space of *value* is not allocated or not enough
KVS_ERR_PARAM_INVALID	*key* or *value* is NULL
KVS_ERR_OFFSET_INVALID	*kvs_value.offset* is invalid
KVS_ERR_OPTION_INVALID	the option is not supported
KVS_ERR_KEY_NOT_EXIST	Key does not exist

6.3.6 kvs_retrieve_kvp_async

kvs_result kvs_retrieve_kvp_async (kvs_key_space_handle ks_hd, kvs_key *key, kvs_option_retrieve *opt, kvs_value *value, kvs_postprocess_function post_fn)

This API asynchronously retrieves a key value pair value with the given key and returns immediately regardless of whether the pair is actually retrieved from a device or not. The final execution results are returned to post process function through kvs_postprocess_context. The value parameter contains output buffer information for the value. As an input value.value contains the buffer to store the key value pair value and value.length contains the buffer size. The key value pair value is copied to value.value buffer and value.length is set to the retrieved value size. If the offset of value is not zero, the value of key value pair is copied into the buffer, skipping the first offset bytes of the value of key value pair. That is, value.length is equal to the total size of (actual_*value_size* – *offset*). The offset is required to align to KVS_ALIGNMENT_UNIT. If the offset is not aligned, a KVS_ERR_VALUE_OFFSET_MISALIGNED error is returned. If an allocated value buffer is not big enough to hold the value, it will set value.actual_value_size to the actual value length and return KVS_ERR_BUFFER_SMALL.

The retrieve option of the retrieve operation is defined in 5.4.8kvs_option_retreive.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key of the key value pair to get value
IN opt	retrieval option. It may be NULL. In that case, the default retrieval option is used.
OUT value	value to receive the key value pair's value from device
IN post_fn	post process function pointer

RETURNS

KVS_SUCCESS to indicate that retrieve is successful or an error code for error.

ERROR CODE

KVS_ERR_VALUE_OFFSET_MISALIGNED	*kvs_value.offset* is not aligned to KVS_ALIGNMENT_UNIT
KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_BUFFER_SMALL	Buffer space of *value* is not allocated or not enough
KVS_ERR_PARAM_INVALID	*key* or *value* is NULL
KVS_ERR_OFFSET_INVALID	*kvs_value.offset* is invalid
KVS_ERR_OPTION_INVALID	the option is not supported
KVS_ERR_KEY_NOT_EXIST	Key does not exist

6.3.7 kvs_store_kvp

kvs_result kvs_store_kvp (kvs_key_space_handle ks_hd, kvs_key *key, kvs_value *value, kvs_option_store *opt)

This API writes a Key-value key value pair into a Key Space. This API supports the modes defined in section 5.4.9 as specified in opt.

Store operations execute based on the existence of the key and the kvs_option_store specified. If the Key Space does not have enough space to store a key value pair, a KVS_ERR_KS_CAPACITY error message is returned.

PARAMETERS

IN ks_hd Key Space handle
IN key Key of the key value pair to store into Key Space
IN value Value of the key value pair to store into Key Space
IN opt Store option. It may be NULL. In that case, the kvs_store_type of *KVS_STORE_POST* (see 5.4.9) is used.

RETURNS

KVS_SUCCESS to indicate that store is successful or an error code for error.

ERROR CODE

KVS_ERR_VALUE_OFFSET_MISALIGNED *kvs_value.offset* is not aligned to KVS_ALIGNMENT_UNIT
KVS_ERR_KS_NOT_EXIST Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID given *key* is not supported (e.g., length)
KVS_ERR_PARAM_INVALID a *key* or a *value* is NULL
KVS_ERR_OFFSET_INVALID *kvs_value.offset* is invalid
KVS_ERR_OPTION_INVALID unsupported option
KVS_ERR_KS_CAPACITY Key Space does not have enough space to store this key value pair
KVS_ERR_VALUE_UPDATE_NOT_ALLOWED a key exists but overwrite is not permitted
KVS_ERR_VALUE_LENGTH_INVALID given value is not supported (e.g., length)

6.3.8 kvs_store_kvp_async

kvs_result kvs_store_kvp_async (kvs_key_space_handle ks_hd, kvs_key *key, kvs_value *value, kvs_option_store *opt, kvs_postprocess_function post_fn)

This API asynchronously writes a Key-value key value pair into a Key Space and returns immediately regardless of whether the pair is actually written to a device or not. The final execution results are returned to post process function through kvs_postprocess_context. This API supports the modes defined in section 5.4.9 .

Store operations execute based on the existence of the key and the kvs_option_store specified. If the Key Space does not have enough space to store a key value pair, a KVS_ERR_KS_CAPACITY error message is returned.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key of the key value pair to store into Key Space
IN value	Value of the key value pair to store into Key Space
IN opt	Store option. It may be NULL. In that case, the kvs_store_type of *KVS_STORE_POST* (see 5.4.9)is used.
IN post_fn	post process function pointer

RETURNS

KVS_SUCCESS to indicate that store is successful or an error code for error.

ERROR CODE

KVS_ERR_VALUE_OFFSET_MISALIGNED	*kvs_value.offset* is not aligned to KVS_ALIGNMENT_UNIT
KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_PARAM_INVALID	a *key* or a *value* is NULL
KVS_ERR_OFFSET_INVALID	*kvs_value.offset* is invalid
KVS_ERR_OPTION_INVALID	unsupported option
KVS_ERR_KS_CAPACITY	Key Space or device does not have enough space to store this key value pair
KVS_ERR_VALUE_UPDATE_NOT_ALLOWED	a key exists but overwrite is not permitted
KVS_ERR_VALUE_LENGTH_INVALID	given value is not supported (e.g., length)

6.3.9 kvs_delete_kvp

kvs_result kvs_delete_kvp (kvs_key_space_handle ks_hd, kvs_key* key, kvs_option_delete *opt)

This API deletes key value pair(s) with a given key.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key of the key value pair(s) to delete
IN opt	delete option

RETURNS

KVS_SUCCESS	Indicate that delete is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*key* is NULL.
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_KEY_NOT_EXIST	*key* does not exist

6.3.10 kvs_delete_kvp_async

kvs_result kvs_delete_kvp_async (kvs_key_space_handle ks_hd, kvs_key* key, kvs_option_delete *opt, kvs_postprocess_function *post_fn)

This API asynchronously deletes key value pair(s) with a given key and returns immediately regardless of whether the pair is actually deleted from a device or not. The final execution results are returned to post process function through kvs_postprocess_context.

PARAMETERS

IN ks_hd	Key Space handle
IN key	Key of the key value pair(s) to delete
IN opt	delete option
IN post_fn	post process function pointer

RETURNS

KVS_SUCCESS	Indicate that delete is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*key* is NULL.
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_KEY_LENGTH_INVALID	given *key* is not supported (e.g., length)
KVS_ERR_KEY_NOT_EXIST	*key* does not exist

6.3.11 kvs_delete_key_group

kvs_result kvs_delete_key_group(kvs_key_space_handle ks_hd,
kvs_key_group_filter   *grp_fltr);

This function call deletes the key-value pairs in a Key Space that matches with *grp_fltr*.

PARAMETERS

IN ks_hd	Key Space handle
IN grp_fltr	Key group filter to delete

RETURNS

KV_SUCCESS to indicate that delete key group is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*grp_fltr* is NULL.
KVS_ERR_SYS_IO	Communication with device failed

6.3.12 kvs_delete_key_group_async

kvs_result kvs_delete_key_group_async(kvs_key_space_handle ks_hd, kvs_key_group_filter *grp_fltr, kvs_postprocess_function post_fn);

This function call deletes the key-value pairs in a Key Space that matches with *grp_fltr* and returns immediately regardless of whether a key group is actually deleted from a device or not. The final execution results are returned to post process function through kvs_postprocess_context.

PARAMETERS

IN ks_hd	Key Space handle
IN grp_fltr	key group filter to delete
IN post_fn	post process function pointer

RETURNS

KV_SUCCESS to indicate that delete key group is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*grp_fltr* is NULL.
KVS_ERR_SYS_IO	Communication with device failed

6.3.13 kvs_exist_kv_pairs

kvs_result kvs_exist_kv_pairs (***kvs_key_space_handle ks_hd***, uint32_t key_cnt, kvs_key *keys, uint32_t buffer_size, kvs_exist_list *list)

This API checks if a set of one or more keys exists and returns a *bool type* status. The existence of a key value pair is determined during an implementation-dependent time window while this API executes. Therefore, repeated routine calls may return different outputs in multi-threaded environments. One bit is used for each key. Therefore when 32 keys are intended to be checked, a caller should allocate 32 bits (i.e., 4 bytes) of memory buffer and the existence information is filled. The LSB (Least Significant Bit) of the *list->result_buffer* indicates if the first key exist or not.

PARAMETERS

IN ks_hd	Key Space handle
IN key_cnt	the number of keys to check
IN keys	a set of keys to check
IN buffer_size	list buffer size in bytes
OUT list	a kvs_exist_list indicates whether corresponding key(s) exists or not

RETURNS

KVS_SUCCESS to indicate success or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_BUFFER_SMALL	the buffer space of list->*result_buffer* is not big enough
KVS_ERR_PARAM_INVALID	*keys* or *list* parameter is NULL
KVS_ERR_SYS_IO	Communication with device failed

6.3.14 kvs_exist_kv_pairs_async

kvs_result kvs_exist_kv_pairs_async(kvs_key_space_handle ks_hd, uint32_t key_cnt, kvs_key *keys, uint32_t buffer_size, kvs_exist_list *list, kvs_postprocess_function post_fn)

This API asynchronously checks if a set of keys exists and returns a *bool type* status. It returns immediately regardless of whether keys are checked from a device or not. The final execution results are returned to the post process function through kvs_postprocess_context. The existence of a key value pair is determined during an implementation-dependent time window while this API executes. Therefore, repeated routine calls is able to return different outputs in multi-threaded environments. One bit is used for each key. Therefore when 32 keys are intended to be checked, a caller shall allocate 32 bits (i.e., 4 bytes) of memory buffer and the existence information is filled. The LSB (Least Significant Bit) of the *list->result_buffer* indicates if the first key exist or not.

PARAMETERS

IN ks_hd	Key Space handle
IN key_cnt	the number of keys
IN keys	a set of keys to check
IN buffer_size	list buffer size in bytes
OUT list	a list indicates whether a corresponding key exists or not
IN post_fn	post process function pointer

RETURNS

KVS_SUCCESS to indicate success or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_BUFFER_SMALL	the buffer space of list->*result_buffer* is not big enough
KVS_ERR_PARAM_INVALID	*keys* or *list* parameter is NULL
KVS_ERR_SYS_IO	Communication with device failed

6.4 Iterator Function calls

6.4.1 kvs_create_iterator

kvs_result kvs_create_iterator(kvs_key_space_handle ks_hd, kvs_option_iterator *iter_op, kvs_key_group_filter *iter_fltr, kvs_iterator_handle *iter_hd)

This function call enables applications to set up a Key Group such that the keys in that Key Group may be iterated within a Key Space (i.e., *kvs_crearte_iterator()* enables a device to prepare a Key Group of keys for iteration by matching a given bit pattern (*it_fltr.bit_pattern)* to all keys in the Key Space considering bits indicated by *it_fltr.bitmask* and the device sets up a Key Group of keys matching that "(*bitmask* & key) == *bit_pattern*".*)* (e.g., if the *bitmask and bit_pattern are 0xF0000000* and *0x30000000* respectively, then *kvs_create_iterator* will prepare a subset of keys which has 0x3XXXXXXX in keys.

Below are some examples of Key Groups.

1) If applications want to get all the existing keys within the device with the first bit of a key set to 1, *kvs_create_iterator()* should be called with *bitmask* = 0x80000000 (1000 0000 0000 0000 0000 0000 0000 0000) and *bit_pattern* = 0x80000000 (1000 0000 0000 0000 0000 0000 0000 0000).
2) If applications want to get all the existing keys within the device with the first bit of key set to 0, *bitmask* should be 0x80000000 (1000 0000 0000 0000 0000 0000 0000 0000) and *bit_pattern* should be 0x0 (0000 0000 0000 0000 0000 0000 0000 0000).
3) If applications want to get all the existing keys with the second and third bytes (bit 8 ~ bit15) equal to 0x04, *bitmask* should be 0x00FF0000 (0000 0000 1111 1111 0000 0000 0000 0000) and *bit_pattern* should be 0x00040000 (0000 0000 0000 0100 0000 0000 0000 0000).
4) If application wants to get all the existing keys with bit 1 ~ bit 4 equal to (0101), *bitmask* should be 0x78000000 (0111 1000 0000 0000 0000 0000 0000 0000) and *bit_pattern* should be 0x28000000 (0010 1000 0000 0000 0000 0000 0000 0000).

It also sets up the iterator option; *kvs_iterator_next()* will only retrieve keys when the kvs_option_iterator is *KVS_ITERATOR_OPT_KEY while kvs_iterator_next()* will retrieve key and value pairs when the kvs_option_iterator is *KVS_ITERATOR_OPT_KV*.
An iterator handle is provided as an output of this function call..

PARAMETERS

IN	ks_hd	Key Space handle
IN	iter_op	iterator option
IN	iter_fltr	iterator filter that includes bitmask and bit pattern
OUT	iter_hd	iterator handle

RETURNS

KVS_SUCCESS to indicate that device open is successful or an error code for error.

ERROR CODE

KVS_ERR_KS_NOT_EXIST Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID *it_fltr* is NULL.
KVS_ERR_SYS_IO Communication with device failed
KVS_ERR_ITERATOR_MAX the maximum number of iterators that a device supports is already open. No more iterator are able to be opened.
KVS_ERR_ITERATOR_OPEN iterator is already opened
KVS_ERR_OPTION_INVALID the device does not support the specified iterator options
KVS_ERR_ITERATOR_FILTER_INVALID iterator filter(match bitmask and pattern) is not valid

6.4.2 kvs_delete_iterator

kvs_result kvs_delete_iterator(kvs_key_space_handle ks_hd, kvs_iterator_handle iter_hd)

This function call releases the resources for the iterator Key Group specified by *iter_hd* in the specified Key Space.

PARAMETERS

IN ks_hd	Key Space handle
IN iter_hd	iterator handle

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_ITERATOR_NOT_EXIST	the iterator Key Group does not exist

6.4.3 kvs_iterate_next

kvs_result kvs_iterate_next(kvs_key_space_handle ks_hd, kvs_iterator_handle iter_hd, uint32_t buffer_size, kvs_iterator_list *iter_list);

This function call obtains a subset of key or key-value pairs from an Key Group of *iter_hd* within a Key Space (i.e., *kvs_iterator_next()* retrieves the next Key Group of keys or key-value pairs in the iterator Key Group (*iter_hd)* that is created with *kvs_create_iterator()* command). *buffer_size* is the iterator buffer (*iter_list*) size in bytes. The retrieved values (*iter_list*) are either keys or key-value pairs based on the iterator option which is specified by *kvs_create_iterator().*

After kvs_create_iterator for a Key Group completes successfully, if a *kvs_store()* or *kvs_delete()* command with a key that matches that Key Group is received, then the keys associated with that command may or may not be included in that iterator.

In the output of this operation, *iter_list.num_entries* provides number of iterator elements in *iter_list.it_list* and *iter_list.end* indicates if there are more elements in the iterator Key Group after this operation. If *iter_list.end* is zero, there are more iterator Key Group elements and the host may run *kvs_iterator_next()* again to retrieve those elements. If *iter_list.end* is one, there are no more iterator Key Group elements and that iterator has reached the last element in the Key Group.

Output values (*iter_list.it_list*) are determined by the iterator option specified by an application.

- **KV_ITERATOR_OPT_KEY [MANDATORY]**: a subset of keys are returned in *iter_list.it_list* data structure
- **KV_ITERATOR_OPT_KEY_VALUE**; a subset of key-value pairs are returned in *iter_list.it_list* data structure

PARAMETERS

IN	ks_hd	Key Space handle
IN	iter_hd	iterator handle
IN	buffer_size	iterator buffer (*iter_list*) size in bytes
OUT	iter_list	output buffer for a set of keys or key-value pairs

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*iter_list* parameter is NULL
KVS_ERR_SYS_IO	Communication with device failed
KVS_ERR_ITERATOR_NOT_EXIST	the iterator Key Group does not exist

6.4.4 kvs_iterate_next_async

kvs_result kvs_iterate_next_async(kvs_key_space_handle ks_hd, kvs_iterator_handle iter_hd , uint32_t buffer_size, kvs_iterator_list *iter_list, kvs_postprocess_function post_fn);

This function call obtains a subset of key or key-value pairs from an iterator Key Group of *iter_hd* within a Key Space (i.e., *kvs_iterator_next()* retrieves a next Key Group of keys or key-value pairs in the iterator key group (*iter_hd)* that is set with *kvs_create_iterator()* command). *buffer_size* is the iterator buffer (*iter_list*) size in bytes. The retrieved values (*iter_list*) are either keys or key-value pairs based on the iterator option which is set by *kvs_create_iterator().*It returns immediately regardless of whether the iterator list is ready from a device or not. The final execution results are returned to the post process function through kvs_postprocess_context.

When *kvs_store()* or *kvs_delete()* command whose key matches with an existing iterator Key Group is received, the keys may or may not be included in the iterator and the inclusion of the updated keys is unspecified.

In the output of this operation, *iter_list.num_entries* provides number of iterator elements in *iter_list.it_list* and *iter_list.end* indicates if there are more elements in the iterator Key Group after this operation. If *iter_list.end* is zero, there are more iterator Key Group elements and host may run *kvs_iterator_next()* again to retrieve those elements. If *iter_list.end* is one, there are no more iterator Key Group elements and the iterator reached the end.

Output values (*iter_list.it_list*) are determined by the iterator option set by an application.

- **KV_ITERATOR_OPT_KEY [MANDATORY]**: a subset of keys are returned in *iter_list.it_list* data structure
- **KV_ITERATOR_OPT_KEY_VALUE**; a subset of key-value pairs are returned in *iter_list.it_list* data structure

PARAMETERS

IN	ks_hd	Key Space handle
IN	iter_hd	iterator handle
IN	buffer_size	iterator buffer (*iter_list*) size in bytes
OUT	iter_list	output buffer for a set of keys or key-value pairs
IN	post_fn	post process function pointer

ERROR CODE

KVS_ERR_KS_NOT_EXIST	Key Space with a given *ks_hd* does not exist
KVS_ERR_PARAM_INVALID	*iter_list* parameter is NULL
KVS_ERR_SYS_IO	Communication with device failed

KVS_ERR_ITERATOR_NOT_EXIST the iterator Key Group does not exist