DaqDB

0.2

Generated by Doxygen 1.8.11

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

DaqDB
Copyright 2017-2018 Intel Corporation . . . . . **??**

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

DaqDB::KVStoreBase
The KVStoreBase is a distributed Key-Value store . . . . . **??**

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

fogkv/daqdb/include/daqdb/**Info.h** . . . . . **??**
fogkv/daqdb/include/daqdb/**Key.h** . . . . . **??**
fogkv/daqdb/include/daqdb/**KVPair.h** . . . . . **??**
fogkv/daqdb/include/daqdb/KVStoreBase.h
Copyright 2017-2018 Intel Corporation . . . . . **??**
fogkv/daqdb/include/daqdb/**Options.h** . . . . . **??**
fogkv/daqdb/include/daqdb/**Status.h** . . . . . **??**
fogkv/daqdb/include/daqdb/**Types.h** . . . . . **??**
fogkv/daqdb/include/daqdb/**Value.h** . . . . . **??**

Chapter 4

Namespace Documentation

4.1 DaqDB Namespace Reference

Classes

- class KVStoreBase

  The KVStoreBase is a distributed Key-Value store.

4.1.1 Detailed Description

Chapter 5

Class Documentation

5.1 DaqDB::KVStoreBase Class Reference

The KVStoreBase is a distributed Key-Value store.

`#include </nfs/igk/disks/innovation/gjerecze/fogkv/daqdb/include/daqdb/KV↩ StoreBase.h>`

Public Types

- using KVStoreBaseCallback = std::function< void(KVStoreBase *kvs, Status status, const char *key, const size_t keySize, const char *value, const size_t valueSize)>

  : jradtke Temporarily using this base alias for both PUT,GET,UPDATE messages

Public Member Functions

- virtual size_t KeySize ()=0

  Return the size of a key, which the KV store uses.
- virtual const Options & getOptions ()=0

  Return the configuration and runtime options which the KV store has been opened with.
- virtual std::string getProperty (const std::string &name)=0

  Return a given property for the KV store.
- virtual void Put (Key &&key, Value &&value, const PutOptions &options=PutOptions())=0

  Synchronously insert a value for a given key.
- virtual void PutAsync (Key &&key, Value &&value, KVStoreBaseCallback cb, const PutOptions &options=PutOptions())=0

  Asynchronously insert a value for a given key.
- virtual Value Get (const Key &key, const GetOptions &options=GetOptions())=0

  Synchronously get a value for a given key.
- virtual Key GetAny (const GetOptions &options=GetOptions())=0

  Synchronously get any unlocked primary key.
- virtual void GetAnyAsync (KVStoreBaseGetAnyCallback cb, const GetOptions &options=GetOptions())=0

  Asynchronously get any unlocked primary key.
- virtual void GetAsync (const Key &key, KVStoreBaseCallback cb, const GetOptions &options=Get↩ Options())=0

Asynchronously get a value for a given key.

- virtual void Update (const Key &key, Value &&value, const UpdateOptions &options=UpdateOptions())=0

  Update value and (optionally) options for a given key.
- virtual void Update (const Key &key, const UpdateOptions &options)=0

  Update options for a given key.
- virtual void UpdateAsync (const Key &key, Value &&value, KVStoreBaseCallback cb, const UpdateOptions &options=UpdateOptions())=0

  Asynchronously update value and (optionally) options for a given key.
- virtual void UpdateAsync (const Key &key, const UpdateOptions &options, KVStoreBaseCallback cb)=0

  Asynchronously update options for a given key.
- virtual std::vector< KVPair > GetRange (const Key &beg, const Key &end, const GetOptions &options=GetOptions())=0

  Synchronously get values for a given range of keys.
- virtual void GetRangeAsync (const Key &beg, const Key &end, KVStoreBaseRangeCallback cb, const GetOptions &options=GetOptions())=0

  Aynchronously get values for a given range of keys.
- virtual void Remove (const Key &key)=0

  Synchronously remove a key-value store entry for a given key.
- virtual void RemoveRange (const Key &beg, const Key &end)=0

  Synchronously remove key-value store entries for a given range of keys.
- virtual Value Alloc (const Key &key, size_t size, const AllocOptions &options=AllocOptions())=0

  Allocate a Value buffer of a given size.
- virtual void Free (Value &&value)=0

  Deallocate a Value buffer.
- virtual void Realloc (Value &value, size_t size, const AllocOptions &options=AllocOptions())=0

  Reallocate a Value buffer.
- virtual void ChangeOptions (Value &value, const AllocOptions &options)=0

  Change allocation options of the given Value buffer.
- virtual Key AllocKey (const AllocOptions &options=AllocOptions())=0

  Allocate a Key buffer.
- virtual void Free (Key &&key)=0

  Deallocate a Key buffer.
- virtual void ChangeOptions (Key &key, const AllocOptions &options)=0

  Change allocation options of the given Key buffer.
- virtual bool IsOffloaded (Key &key)=0

  CHecks if given key is moved to long term storage.

Static Public Member Functions

- static KVStoreBase * Open (const Options &options)

  Open the KV store with given options.

5.1.1 Detailed Description

The KVStoreBase is a distributed Key-Value store.

The key size is constant and the value is variadic length non-null terminated stream of bytes.

This is a non-template class version of KV store, in which both key and value are represented by (char *) type.

This class is safe for concurrent use from multiple threads without a need for external synchronization.

5.1.2 Member Function Documentation

5.1.2.1 virtual Value DaqDB::KVStoreBase::Alloc (const Key & *key,* size_t *size,* const AllocOptions & *options* = AllocOptions()) [pure virtual]

Allocate a Value buffer of a given size.

Returns

On success returns a pointer to allocated KV buffer. Otherwise returns nullptr.

Parameters

in	*size*	Size of allocation.
in	*options*	Allocation options.

5.1.2.2 virtual Key DaqDB::KVStoreBase::AllocKey (const AllocOptions & *options* = AllocOptions()) [pure virtual]

Allocate a Key buffer.

Returns

On success returns a pointer to allocated Key buffer. Otherwise returns nullptr.

Parameters

in	*options*	Allocation options.

5.1.2.3 virtual void DaqDB::KVStoreBase::ChangeOptions (Value & *value,* const AllocOptions & *options*) [pure virtual]

Change allocation options of the given Value buffer.

Parameters

in	*value*	Value buffer.
in	*options*	Allocation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.4 virtual void DaqDB::KVStoreBase::ChangeOptions (Key & *key*, const AllocOptions & *options*) [pure virtual]

Change allocation options of the given Key buffer.

Parameters

in	*key*	Key buffer.
in	*options*	Allocation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.5 virtual void DaqDB::KVStoreBase::Free (Value && *value*) [pure virtual]

Deallocate a Value buffer.

Parameters

in	*value*	Value buffer. If not allocated by current instance of KVStoreBase the behaviour is undefined.

5.1.2.6 virtual void DaqDB::KVStoreBase::Free (Key && *key*) [pure virtual]

Deallocate a Key buffer.

Parameters

in	*key*	Key buffer. If not allocated by current instance of KVStoreBase the behavior is undefined.

5.1.2.7 virtual Value DaqDB::KVStoreBase::Get (const Key & *key*, const GetOptions & *options* = GetOptions()) [pure virtual]

Synchronously get a value for a given key.

Returns

On success returns allocated buffer with value. The caller is responsible of releasing the buffer.

Parameters

in	*key*	Reference to a key structure.
in	*options*	Get operation options.

Exceptions

OperationFailedException	if any error occurred XXX

```
struct Key {
    Key() = default;
    Key(uint64_t e, uint16_t s, uint16_t r)
        : event_id(e), subdetector_id(s), run_id(r) {}
    uint64_t event_id;
    uint16_t subdetector_id;
    uint16_t run_id;
};
```

```
    try {

        DaqDB::Key key = kvs->AllocKey();

        // Fill the key
        Key *keyp = reinterpret_cast<Key *>(key.data());
        keyp->subdetector_id = 1;
        keyp->run_id = 2;
        keyp->event_id = 3;

        // Get operation, the caller becomes the owner of a local copy
        // of the value. The caller must free both key and value buffers by
        // itself, or
        // transfer the ownership in another call.
        DaqDB::Value value;
        try {
            value = kvs->Get(key);
        } catch (...) {
            // error
        }

        // success, process the data and free the buffers
        kvs->Free(std::move(key));
        kvs->Free(std::move(value));
    } catch (...) {
        // error
    }
```

5.1.2.8 virtual Key DaqDB::KVStoreBase::GetAny (const GetOptions & *options* = `GetOptions()`) [pure virtual]

Synchronously get any unlocked primary key.

Other fields of the key are invalid.

Returns

On success returns a Key of an unprocessed key.

Parameters

in	*options*	Operation options.

```
struct Key {
    Key() = default;
    Key(uint64_t e, uint16_t s, uint16_t r)
        : event_id(e), subdetector_id(s), run_id(r) {}
```

```
    uint64_t event_id;
    uint16_t subdetector_id;
    uint16_t run_id;
};
```

```
    DaqDB::Options options;

    // Configure key structure
    options.Key.field(0, sizeof(Key::event_id), true); // primary key
    options.Key.field(1, sizeof(Key::subdetector_id));
    options.Key.field(2, sizeof(Key::run_id));

    DaqDB::KVStoreBase *kvs;
    try {
        kvs = DaqDB::KVStoreBase::Open(options);
    } catch (DaqDB::OperationFailedException &e) {
        // error
        // e.status()
    }

    // success
```

```
    try {

        DaqDB::GetOptions getOptions;
        getOptions.Attr = DaqDB::READY;
        getOptions.NewAttr = DaqDB::LOCKED | DaqDB::READY;

        // get and lock any primary key which is in unlocked state
        DaqDB::Key keyBuff = kvs->GetAny(getOptions);
        Key *key = reinterpret_cast<Key *>(keyBuff.data());

        Key keyBeg(key->event_id, std::numeric_limits<uint16_t>::min(),
                   std::numeric_limits<uint16_t>::min());

        Key keyEnd(key->event_id, std::numeric_limits<uint16_t>::max(),
                   std::numeric_limits<uint16_t>::max());

        std::vector<DaqDB::KVPair> range = kvs->GetRange(
            DaqDB::Key(reinterpret_cast<char *>(&keyBeg), sizeof(keyBeg)),
            DaqDB::Key(reinterpret_cast<char *>(&keyEnd), sizeof(keyEnd)));

        for (auto kv : range) {
            // or unlock and mark the primary key as ready
            kvs->Update(kv.key(), DaqDB::READY);
        }

    } catch (...) {
        // error
    }
```

5.1.2.9 virtual void DaqDB::KVStoreBase::GetAnyAsync (KVStoreBaseGetAnyCallback *cb*, const GetOptions & *options* = GetOptions()) [pure virtual]

Asynchronously get any unlocked primary key.

Other fields of the key are invalid.

Parameters

in	*options*	Operation options.
in	*cb*	Callback function. Will be called when the operation completes.

5.1.2.10 virtual void DaqDB::KVStoreBase::GetAsync (const Key & *key*, KVStoreBaseCallback *cb*, const GetOptions & *options* = GetOptions()) [pure virtual]

Asynchronously get a value for a given key.

Parameters

in	*key*	Reference to a key structure.
in	*cb*	Callback function. Will be called when the operation completes with results passed in arguments.
in	*options*	Get operation options.

Exceptions

OperationFailedException	if any error occurred XXX

```
struct Key {
    Key() = default;
    Key(uint64_t e, uint16_t s, uint16_t r)
        : event_id(e), subdetector_id(s), run_id(r) {}
    uint64_t event_id;
    uint16_t subdetector_id;
    uint16_t run_id;
};
```

```
    try {
        DaqDB::Key key = kvs->AllocKey();

        // Fill the key
        Key *keyp = reinterpret_cast<Key *>(key.data());
        keyp->subdetector_id = 1;
        keyp->run_id = 2;
        keyp->event_id = 3;

        try {
            kvs->GetAsync(key,
                          [&](DaqDB::KVStoreBase *kvs, DaqDB::Status status,
                              const char *key, size_t keySize,
                              const char *value, size_t valueSize) {

                              if (!status.ok()) {
                                  // error
                                  return;
                              }

                              // process value

                              // free the value buffer
                          });
        } catch (DaqDB::OperationFailedException &exc) {
            // error, status in:
            // exc.status();
            kvs->Free(std::move(key));
        }
    } catch (...) {
        // error
    }
```

5.1.2.11 virtual const Options& DaqDB::KVStoreBase::getOptions ( ) [pure virtual]

Return the configuration and runtime options which the KV store has been opened with.

Returns

Configuration and runtime options

5.1.2.12 virtual std::string DaqDB::KVStoreBase::getProperty (const std::string & *name*) [pure virtual]

Return a given property for the KV store.

XXX full documentation of supported properties

5.1.2.13 virtual std::vector<KVPair> DaqDB::KVStoreBase::GetRange (const Key & *beg,* const Key & *end,* const GetOptions & *options* = GetOptions()) [pure virtual]

Synchronously get values for a given range of keys.

Returns

On success returns Ok, on failure returns a value indicating an error occurred.

Parameters

in	*beg*	key representing the beginning of a range.
in	*end*	key representing the end of a range.
in	*options*	Get operation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.14 virtual void DaqDB::KVStoreBase::GetRangeAsync (const Key & *beg,* const Key & *end,* KVStoreBaseRangeCallback *cb,* const GetOptions & *options* = GetOptions()) [pure virtual]

Aynchronously get values for a given range of keys.

Parameters

in	*beg*	key representing the beginning of a range.
in	*end*	key representing the end of a range.
in	*cb*	Callback function. Will be called when the operation completes with results passed in arguments.
in	*options*	Get operation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.15 virtual bool DaqDB::KVStoreBase::IsOffloaded (Key & *key*) [pure virtual]

CHecks if given key is moved to long term storage.

Parameters

in	*key*	Key buffer.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.16 virtual size_t DaqDB::KVStoreBase::KeySize ( ) [pure virtual]

Return the size of a key, which the KV store uses.

Returns

Key size in bytes

5.1.2.17 static KVStoreBase* DaqDB::KVStoreBase::Open (const Options & *options*) [static]

Open the KV store with given options.

Returns

On success, returns a pointer to a heap-allocated KV Store otherwise returns nullptr

Parameters

in	*options*	Required options parameter which contains KV Store configuration and runtime options.

```
DaqDB::Options options;

// Configure key structure
options.Key.field(0, sizeof(Key::event_id), true); // primary key
options.Key.field(1, sizeof(Key::subdetector_id));
options.Key.field(2, sizeof(Key::run_id));

DaqDB::KVStoreBase *kvs;
try {
    kvs = DaqDB::KVStoreBase::Open(options);
} catch (DaqDB::OperationFailedException &e) {
    // error
    // e.status()
}

// success
```

5.1.2.18 virtual void DaqDB::KVStoreBase::Put (Key && *key*, Value && *value*, const PutOptions & *options* = PutOptions()) [pure virtual]

Synchronously insert a value for a given key.

Note

The ownership of key and value buffers are transferred to the KVStoreBase object.

Parameters

in	*key*	Rvalue reference to key buffer.
in	*value*	Rvalue reference to value buffer
in	*options*	Put operation options.

Exceptions

OperationFailedException	if any error occurred XXX

```
struct Key {
    Key() = default;
    Key(uint64_t e, uint16_t s, uint16_t r)
        : event_id(e), subdetector_id(s), run_id(r) {}
    uint64_t event_id;
    uint16_t subdetector_id;
    uint16_t run_id;
};
```

```
    try {
        DaqDB::Key key = kvs->AllocKey();

        // Fill the key
        Key *keyp = reinterpret_cast<Key *>(key.data());
        keyp->subdetector_id = 1;
        keyp->run_id = 2;
        keyp->event_id = 3;

        DaqDB::Value value = kvs->Alloc(key, 1024);

        // Fil the value buffer
        std::memset(value.data(), 0, value.size());

        // Put operation, transfer ownership of key and value buffers to library
        kvs->Put(std::move(key), std::move(value));
    } catch (...) {
        // error
    }
```

5.1.2.19 virtual void DaqDB::KVStoreBase::PutAsync (Key && *key*, Value && *value*, KVStoreBaseCallback *cb*, const PutOptions & *options* = `PutOptions()`) [pure virtual]

Asynchronously insert a value for a given key.

Note

The ownership of key and value buffers are transferred to the KVStoreBase object.

Parameters

in	*key*	Rvalue reference to key buffer.
in	*value*	Rvalue reference to value buffer
in	*cb*	Callback function. Will be called when the operation completes with results passed in arguments.
in	*options*	Put operation options.

Exceptions

OperationFailedException	if any error occurred XXX

```
struct Key {
    Key() = default;
    Key(uint64_t e, uint16_t s, uint16_t r)
        : event_id(e), subdetector_id(s), run_id(r) {}
    uint64_t event_id;
    uint16_t subdetector_id;
    uint16_t run_id;
};
```

```
    try {
        DaqDB::Key key = kvs->AllocKey();

        // Fill the key
        Key *keyp = reinterpret_cast<Key *>(key.data());
        keyp->subdetector_id = 1;
        keyp->run_id = 2;
        keyp->event_id = 3;

        DaqDB::Value value = kvs->Alloc(key, 1024);

        // Fill the value buffer
        std::memset(value.data(), 0, value.size());

        // Asynchronous Put operation, transfer ownership of key and value
        // buffers to library
        kvs->PutAsync(std::move(key), std::move(value),
                      [&](DaqDB::KVStoreBase *kvs, DaqDB::Status status,
                          const char *key, size_t keySize, const char *value,
                          size_t valueSize) {
                          if (!status.ok()) {
                              // error
                              return;
                          }

                      });
    } catch (...) {
        // error
    }
```

5.1.2.20 virtual void DaqDB::KVStoreBase::Realloc (Value & *value*, size_t *size*, const AllocOptions & *options* = AllocOptions()) [pure virtual]

Reallocate a Value buffer.

Parameters

in	*value*	KV buffer to be reallocated. If buff is nullptr this call is equivalent to Alloc.
in	*size*	New size of a Value buffer. If the size is 0 and buff is not nullptr, this call is equivalent to Free.
in	*options*	New options for a Value buffer.

Exceptions

OperationFailedException	if any error occurred XXX

Note

It is possible to modify the options of an allocation without changing a size, by passing the same size.

5.1.2.21 virtual void DaqDB::KVStoreBase::Remove (const Key & *key*) [pure virtual]

Synchronously remove a key-value store entry for a given key.

Parameters

in	*key*	Pointer to a key structure.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.22 virtual void DaqDB::KVStoreBase::RemoveRange (const Key & *beg*, const Key & *end*) [pure virtual]

Synchronously remove key-value store entries for a given range of keys.

Parameters

in	*beg*	Pointer to a key structure representing the beginning of a range.
in	*end*	Pointer to a key structure representing the end of a range.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.23 virtual void DaqDB::KVStoreBase::Update (const Key & *key*, Value && *value*, const UpdateOptions & *options* = UpdateOptions()) [pure virtual]

Update value and (optionally) options for a given key.

Note

The ownership of the value buffer is transferred to the KVStoreBase object.

Parameters

in	*key*	Reference to a key buffer.
in	*value*	Rvalue reference to a value buffer.
in	*options*	Update operation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.24 virtual void DaqDB::KVStoreBase::Update (const Key & *key*, const UpdateOptions & *options*) [pure virtual]

Update options for a given key.

Parameters

in	*key*	Reference to a key buffer.
in	*options*	Update operation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.25 virtual void DaqDB::KVStoreBase::UpdateAsync (const Key & *key*, Value && *value*, KVStoreBaseCallback *cb*, const UpdateOptions & *options* = UpdateOptions()) [pure virtual]

Asynchronously update value and (optionally) options for a given key.

Note

The ownership of the value buffer is transferred to the KVStoreBase object.

Parameters

in	*key*	Reference to a key buffer.
in	*value*	Rvalue reference to a value buffer.
in	*cb*	Callback function. Will be called when the operation completes with results passed in arguments.
in	*options*	Update operation options.

Exceptions

OperationFailedException	if any error occurred XXX

5.1.2.26 virtual void DaqDB::KVStoreBase::UpdateAsync (const Key & *key*, const UpdateOptions & *options*, KVStoreBaseCallback *cb*) [pure virtual]

Asynchronously update options for a given key.

Note

The ownership of the value buffer is transferred to the KVStoreBase object.

Parameters

in	*key*	Reference to a key buffer.
in	*cb*	Callback function. Will be called when the operation completes with results passed in arguments.
in	*options*	Update operation options.

Exceptions

OperationFailedException	if any error occurred XXX

The documentation for this class was generated from the following file:

- fogkv/daqdb/include/daqdb/KVStoreBase.h

Chapter 6

File Documentation

6.1 fogkv/daqdb/include/daqdb/KVStoreBase.h File Reference

Copyright 2017-2018 Intel Corporation.

```
#include <string>
#include "daqdb/Status.h"
#include "daqdb/Options.h"
#include "daqdb/Key.h"
#include "daqdb/Value.h"
#include "daqdb/KVPair.h"
#include <functional>
```

Classes

- class DaqDB::KVStoreBase

  The KVStoreBase is a distributed Key-Value store.

Namespaces

- DaqDB

  Copyright 2017-2018 Intel Corporation.

6.1.1 Detailed Description

KVStoreBase the base class of Key-Value store.