Key-Value Store Developer Manual

1 Adding a new key-value store

To add a new key-value store, we must implement the KVImplHelper class form KVImpl.h header file, present in KeyValueStore/Interface directory. This class is used to implement the functionality of the classes KVStore and KVRequest from KVStoreHeader.h. We must implement the following functionality of KVImplHelper class.

- **bool bind(string connection,string tablename):**
  This function takes in the connection string and table name as parameter and is supposed to initialize the connection to the key-value store. It must return true if connection to key-value store was successfully established, otherwise false.

- **KVData<string> get(string const& key):**
  This function takes in the key as string and is supposed to fetch the value corresponding to the key from the key-value store. The table used here is the one bound during the bind operation. If an error occurs during the get operation then it must be notified by setting the value of ierr field of KVData object to some negative value and a human readable error in serr field of KVData object. If the value is retrieved without any error, set the value of ierr to zero and serr blank, set value field of KVData to the retrieved value and return the KVData object.

- **KVData<string> put(string const& key,string const& val):**
  This function takes in the key and value as string and is supposed to store the key-value pair in the key-value store. The table used here is the one bound during the bind operation. If an error occurs during the put operation then it must be notified by setting the value of ierr field of KVData object to some negative value and a human readable error in serr field of KVData object. If the operation was successful then set the value of ierr to zero and serr blank.

- **KVData<string> del(string const& key):**
  This function takes in the key as string and is supposed to delete the key-value from the key-value store. The table used here is the one bound during the bind operation. If an error occurs during the get operation then it must be notified by setting the value of ierr field of KVData object to some negative value and a human readable error in serr field of KVData object. If the value was successfully deleted without any error, set the value of ierr to zero and serr blank.

- **void async_get(string key, void (*fn)(KVData<string>, void *), void *vfn):**
  This is the asynchronous version of get function, it must not wait for the get request to complete and must immediately return the control to calling thread once the request has been registered. When the request gets completed, it must call the function provided in "fn" parameter with the response as its first parameter and "vnf" as the second parameter.

- **void async_put(string key, string val, void (*fn)(KVData<string>, void *), void *vfn):**
  This is the asynchronous version of put function, similarly it must not wait for the put request to complete and must immediately return the control to calling thread once the request has been registered. When the request gets completed, it must call the function provided in "fn" parameter with the response as its first parameter and "vnf" as the second parameter.

- **void async_del(string key, void (*fn)(KVData<string>,void *), void *vfn):**
  Similarly, this is the asynchronous version of del function and it must not wait for the del request to complete and must return the control to the calling thread. When the request gets completed, it must call the function provided in "fn" parameter with the response as its first parameter and "vnf" as the second parameter.

- **bool clear():**
  This function is supposed to delete all the content from the table associated with KVImplHelper object and must return true if the operation was successful and false otherwise.

- **int mget(vector<string>& key, vector<string>& tablename, vector<KVData<string>>& ret):**

This is the multi get version of get function, this function is supposed to fetch multiple values from multiple tables. The input here is a vector of keys and a corresponding vector of table names. The response must be provided by appending the responses to the "ret" vector.

- **int mdel(vector<string>& key, vector<string>& tablename, vector<KVData<string>>& ret):**
  Similarly, this is the multi del version of del function, this function is supposed to delete given vector of keys from the corresponding tables.Here also the response must be provided by appending the responses to the "ret" vector.

- **int mput(vector<string>& key, vector<string>& val, vector<string>& tablename, vector<KVData<string>>& ret):**
  This is the multi put version of put function, this function is supposed to store the given set of keys and values to their corresponding tables.Here also the response must be provided by appending the responses to the "ret" vector.

- **void async_get(string key, string tablename, void (*fn)(KVData<string>, void *), void *vfn):**
  This is another version of async_get and does the same task async_get, but instead of getting the value from the table associated with KVImpleHelper object, it fetches the value from the table identified by the parameter tablename.

- **void async_put(string key, string val, string tablename, void (*fn)(KVData<string>, void *), void *vfn):**
  Similarly this version of async_put is same as previous async_put but it operates on the given table.

- **void async_del(string key, string tablename, void (*fn)(KVData<string>, void *), void *vfn):**
  Similarly this version of async_del is same as previous async_del but it operates on the given table.

- **int smget(vector<string>& key, vector<string>& tablename, vector<KVData<string>>& ret):**
  This is the safe version of mget, and it must store the state required by smput to carry out its operation.

- **int smput(vector<string>& key, vector<string>& val, vector<string>& tablename, vector<KVData<string>>& ret):**

This is the safe version of mput, but it ensures that the values updated by it are not modified in between the last smget of that key and up to its update by this smput operation. Smput for a particular key must fail if the value has been modified in between the last smget of that key and this update.

- **void *dataholder:**
  It is a private member of class KVImplHelper of type void pointer, and can be used to store any generic structure useful for implementation.

For examples of KVImpleHelper implementation please look into the "KeyValueStore/Implementation/(any data store)/client/src/"

2 Useful Links

- **LevelDB:**
  https://github.com/google/leveldb
  https://github.com/syndtr/goleveldb/tree/master/leveldb

- **Memcached:**
  https://memcached.org/
  http://libmemcached.org/libMemcached.html
  http://docs.libmemcached.org/
  http://docs.libmemcached.org/genindex.html

- **RAMCloud:**
  https://ramcloud.atlassian.net/wiki/display/RAM/RAMCloud
  https://ramcloud.atlassian.net/wiki/display/RAM/General+Information+for+Developers
  https://ramcloud.stanford.edu/docs/doxygen/hierarchy.html

- **Redis:**
  https://redis.io/
  https://redis.io/documentation
  https://redis.io/topics/cluster-tutorial
  https://github.com/vipshop/hiredis-vip
  https://github.com/redis/hiredis