BLINK DB - Part A

24CS60R77 Ishan Rai

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 HashTable< K, V > Class Template Reference

Custom hash table implementation with separate chaining collision resolution.

`#include <HashTable.h>`

Public Member Functions

- std::vector< Node * > & get_table ()
  Get the internal table structure.
- size_t get_capacity () const
  Get current table capacity.
- HashTable (size_t initial_capacity=8)
  Construct a new Hash Table object.
- ~HashTable ()
  Destroy the Hash Table object.
- void insert (const K &key, const V &value)
  Insert or update a key-value pair.
- bool get (const K &key, V &value) const
  Retrieve value for a key.
- bool remove (const K &key)
  Remove a key-value pair.
- void clear ()
  Clear all entries from the hash table.
- HashTable (const HashTable &)=delete
  Disabled copy constructor.
- HashTable & operator= (const HashTable &)=delete
  Disabled assignment operator.

3.1.1 Detailed Description

template<typename K, typename V>
class HashTable< K, V >

Custom hash table implementation with separate chaining collision resolution.

Template Parameters

K	Key type
V	Value type

Implements a hash table with dynamic resizing and LRU-friendly structure. Uses separate chaining for collision resolution. Automatically resizes when load factor exceeds 0.7 or falls below 0.2 (for capacities > 8).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 HashTable() [1/2]

```
template<typename K , typename V >
HashTable< K, V >::HashTable (
            size_t initial_capacity = 8 )  [inline], [explicit]
```

Construct a new Hash Table object.

Parameters

initial_capacity	Starting number of buckets (default: 8)

Exceptions

std::invalid_argument	If initial_capacity < 1

3.1.2.2 ~HashTable()

```
template<typename K , typename V >
HashTable< K, V >::~HashTable ( )  [inline]
```

Destroy the Hash Table object.

Clears all nodes and deallocates memory

3.1.2.3 HashTable() [2/2]

```
template<typename K , typename V >
HashTable< K, V >::HashTable (
            const HashTable< K, V > &  )  [delete]
```

Disabled copy constructor.

3.1.3 Member Function Documentation

3.1.3.1 clear()

```
template<typename K , typename V >
void HashTable< K, V >::clear ( )  [inline]
```

Clear all entries from the hash table.

Deallocates all nodes and resets to initial state

3.1.3.2 get()

```
template<typename K , typename V >
bool HashTable< K, V >::get (
            const K & key,
            V & value ) const  [inline]
```

Retrieve value for a key.

Parameters

key	The key to search for
value[out]	Reference to store retrieved value

Returns

true If key was found

false If key was not found

3.1.3.3 get_capacity()

```
template<typename K , typename V >
size_t HashTable< K, V >::get_capacity ( ) const  [inline]
```

Get current table capacity.

Returns

size_t Number of buckets

3.1.3.4 get_table()

```
template<typename K , typename V >
std::vector< Node * > & HashTable< K, V >::get_table ( )  [inline]
```

Get the internal table structure.

Returns

std::vector<Node*>& Reference to the bucket array

3.1.3.5 insert()

```
template<typename K , typename V >
void HashTable< K, V >::insert (
            const K & key,
            const V & value )  [inline]
```

Insert or update a key-value pair.

Parameters

key	The key to insert/update
value	The value to associate with the key

Automatically resizes table if load factor exceeds threshold. Time complexity: O(1) average case, O(n) worst case

3.1.3.6 operator=()

```
template<typename K , typename V >
HashTable & HashTable< K, V >::operator= (
            const HashTable< K, V > &  )  [delete]
```

Disabled assignment operator.

3.1.3.7 remove()

```
template<typename K , typename V >
bool HashTable< K, V >::remove (
            const K & key )  [inline]
```

Remove a key-value pair.

Parameters

key	The key to remove

Returns

true If key was found and removed

false If key was not found

Automatically shrinks table if load factor falls below 0.2 (for capacities > 8)

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

- src/HashTable.h

3.2 MemoryManager Class Reference

Manages Least Recently Used (LRU) eviction policy.

`#include <MemoryManager.h>`

Public Member Functions

- void update_lru (const std::string &key)
  Update LRU queue on key access.
- std::string evict_lru ()
  Evict least recently used key.
- void evict_lru (const std::string &key)
  Remove specific key from LRU tracking.

3.2.1 Detailed Description

Manages Least Recently Used (LRU) eviction policy.

Tracks key access patterns using a queue structure. Integrates with StorageEngine to enforce memory limits as per project requirements.

3.2.2 Member Function Documentation

3.2.2.1 evict_lru() [1/2]

```
std::string MemoryManager::evict_lru ( )  [inline]
```

Evict least recently used key.

Returns

std::string Evicted key (empty if queue is empty)

Removes and returns the LRU key from queue back. Complexity O(1). Called when enforcing memory limits.

3.2.2.2 evict_lru() [2/2]

```
void MemoryManager::evict_lru (
            const std::string & key )  [inline]
```

Remove specific key from LRU tracking.

Parameters

key	The key to remove

Used when keys are explicitly deleted. Complexity O(n). Maintains queue consistency after manual deletions.

3.2.2.3 update_lru()

```
void MemoryManager::update_lru (
            const std::string & key )  [inline]
```

Update LRU queue on key access.

Parameters

key	The accessed key

Moves the key to the front of the LRU queue. Complexity O(n) due to list search. Called on GET/SET operations.

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

- src/MemoryManager.h

3.3 REPL Class Reference

Command processor for BLINK DB storage engine.

`#include <REPL.h>`

Public Member Functions

- REPL (StorageEngine &engine)
  Construct a new REPL interface.
- void process_command (const std::string &input)
  Process a single user command.

3.3.1 Detailed Description

Command processor for BLINK DB storage engine.

Handles parsing and execution of user commands according to the specification:

- SET <key> "<value>" [EX <seconds>]
- GET <key>
- DEL <key>

Maintains strict separation from StorageEngine implementation (Note 2 compliance)

3.3.2 Constructor & Destructor Documentation

3.3.2.1 REPL()

```
REPL::REPL (
            StorageEngine & engine )  [inline]
```

Construct a new REPL interface.

Parameters

engine	Reference to StorageEngine instance

3.3.3 Member Function Documentation

3.3.3.1 process_command()

```
void REPL::process_command (
            const std::string & input )  [inline]
```

Process a single user command.

Parameters

input	Raw command string from user

Implements full command processing workflow:

1. Tokenization of input
2. Command validation
3. Execution via StorageEngine
4. Error handling and output formatting

Exceptions

std::invalid_argument	For invalid TTL values
std::exception	For general processing errors

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

- src/REPL.h

3.4 StorageEngine Class Reference

In-memory key-value store with LRU eviction and TTL expiration.

```
#include <StorageEngine.h>
```

Public Member Functions

- StorageEngine (size_t max_memory=1024 ∗1024 ∗1024)

  Construct a new Storage Engine.
- ∼StorageEngine ()

  Destroy the Storage Engine.
- void set (const std::string &key, const std::string &value, std::chrono::seconds ttl=std::chrono::seconds::max())

  Store/update a key-value pair.
- std::string get (const std::string &key)

  Retrieve value for key.
- bool del (const std::string &key)

  Delete a key-value pair.

3.4.1 Detailed Description

In-memory key-value store with LRU eviction and TTL expiration.

Implements CRUD operations using custom HashTable and MemoryManager. Designed for Part 1 of DESIGN_LAB_PROJECT.pdf specifications with:

- O(1) average case performance
- Thread-safe operations
- Background TTL eviction thread

3.4.2 Constructor & Destructor Documentation

3.4.2.1 StorageEngine()

```
StorageEngine::StorageEngine (
            size_t max_memory = 1024 * 1024 * 1024 )  [explicit]
```

Construct a new Storage Engine.

Construct a new Storage Engine object.

Parameters

max_memory	Maximum allowed memory in bytes (default: 1GB)
max_memory	Maximum allowed memory in bytes

Initializes the storage engine with specified memory limit and starts background eviction daemon thread

3.4.2.2 ∼StorageEngine()

```
StorageEngine::~StorageEngine ( )
```

Destroy the Storage Engine.

Destroy the Storage Engine object.

Stops eviction thread and cleans up resources

3.4.3 Member Function Documentation

3.4.3.1 del()

```
bool StorageEngine::del (
            const std::string & key )
```

Delete a key-value pair.

Parameters

key	Key to remove

Returns

true If key existed and was deleted
false If key didn't exist

Note

Thread-safe through mutex locking

Parameters

key	Key to delete

Returns

true If key existed and was deleted
false If key didn't exist

Implements DEL operation with:

- Memory usage adjustment
- LRU tracking cleanup

Note

Locks mutex during operation

3.4.3.2 get()

```
std::string StorageEngine::get (
            const std::string & key )
```

Retrieve value for key.

Retrieve value for a key.

Parameters

key	Key to lookup

Returns

std::string Value or empty string if not found/expired

Updates last_accessed timestamp for LRU tracking

Note

Thread-safe through mutex locking

Parameters

key	Key to look up

Returns

std::string Retrieved value or empty string

Implements GET operation with:

- Access time updating
- TTL expiration checks
- LRU tracking updates

Note

Locks mutex during operation

3.4.3.3 set()

```
void StorageEngine::set (
            const std::string & key,
            const std::string & value,
            std::chrono::seconds ttl = std::chrono::seconds::max() )
```

Store/update a key-value pair.

Store or update a key-value pair.

Parameters

key	Unique identifier
value	Data to store
ttl	Time-to-live in seconds (default: no expiration)

Note

Thread-safe through mutex locking

Parameters

key	Key to store/update
value	Value to associate with key
ttl	Time-to-live in seconds (default: no expiration)

Implements SET operation with thread safety and memory management:

- Updates existing entries' memory usage
- Applies LRU tracking
- Enforces memory limits

Note

Locks mutex during operation

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

- src/StorageEngine.h
- src/StorageEngine.cpp

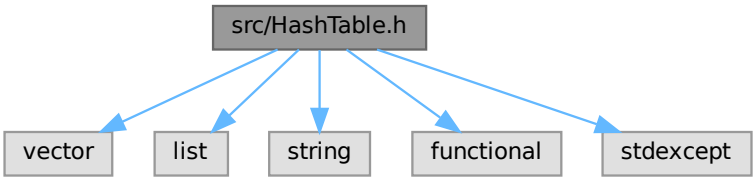
Chapter 4

File Documentation

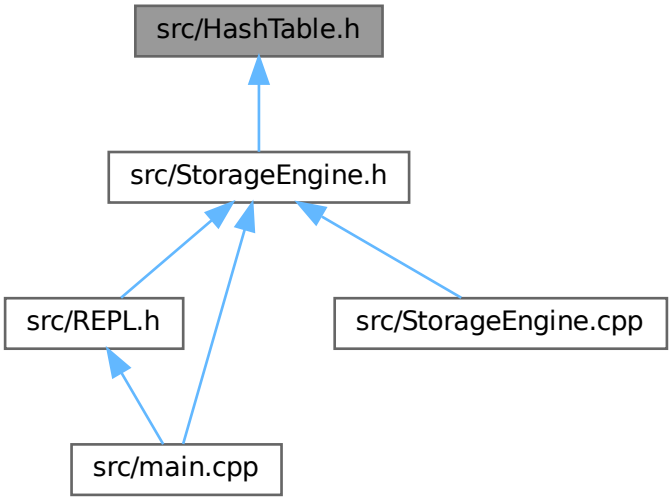
4.1 src/HashTable.h File Reference

```
#include <vector>
#include <list>
#include <string>
#include <functional>
#include <stdexcept>
```

Include dependency graph for HashTable.h:



This graph shows which files directly or indirectly include this file:



Classes

- class HashTable< K, V >

  Custom hash table implementation with separate chaining collision resolution.

4.2 HashTable.h

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <vector>
00003 #include <list>
00004 #include <string>
00005 #include <functional>
00006 #include <stdexcept>
00007
00018 template <typename K, typename V>
00019 class HashTable
00020 {
00021 private:
00026     struct Node
00027     {
00028         K key;
00029         V value;
00030         Node *next;
00031
00038         Node(const K &k, const V &v, Node *n = nullptr)
00039             : key(k), value(v), next(n) {}
00040     };
00041
00042     std::vector<Node *> table;
00043     size_t capacity;
00044     size_t size;
00045     const double LOAD_FACTOR = 0.7;
00046
00052     size_t hash(const K &key) const
00053     {
00054         return std::hash<K>{}(key) % capacity;
00055     }
```

```
00056
00063     void resize(size_t new_capacity)
00064     {
00065         std::vector<Node *> new_table(new_capacity, nullptr);
00066
00067         // Rehash all entries
00068         for (size_t i = 0; i < capacity; ++i)
00069         {
00070             Node *current = table[i];
00071             while (current)
00072             {
00073                 Node *next = current->next;
00074                 size_t new_index = std::hash<K>{}(current->key) % new_capacity;
00075
00076                 current->next = new_table[new_index];
00077                 new_table[new_index] = current;
00078
00079                 current = next;
00080             }
00081         }
00082
00083         table = std::move(new_table);
00084         capacity = new_capacity;
00085     }
00086
00087 public:
00092     std::vector<Node *> &get_table() { return table; }
00093
00098     size_t get_capacity() const { return capacity; }
00099
00105     explicit HashTable(size_t initial_capacity = 8)
00106         : capacity(initial_capacity), size(0)
00107     {
00108         if (initial_capacity < 1)
00109             throw std::invalid_argument("Invalid capacity");
00110         table.resize(capacity, nullptr);
00111     }
00112
00117     ~HashTable()
00118     {
00119         clear();
00120     }
00121
00130     void insert(const K &key, const V &value)
00131     {
00132         if (size >= LOAD_FACTOR * capacity)
00133         {
00134             resize(2 * capacity);
00135         }
00136
00137         size_t index = hash(key);
00138         Node *current = table[index];
00139
00140         // Update existing key if found
00141         while (current)
00142         {
00143             if (current->key == key)
00144             {
00145                 current->value = value;
00146                 return;
00147             }
00148             current = current->next;
00149         }
00150
00151         // Insert new node at head of chain
00152         table[index] = new Node(key, value, table[index]);
00153         size++;
00154     }
00155
00163     bool get(const K &key, V &value) const
00164     {
00165         size_t index = hash(key);
00166         Node *current = table[index];
00167
00168         while (current)
00169         {
00170             if (current->key == key)
00171             {
00172                 value = current->value;
00173                 return true;
00174             }
00175             current = current->next;
00176         }
00177         return false;
00178     }
00179
00189     bool remove(const K &key)
```

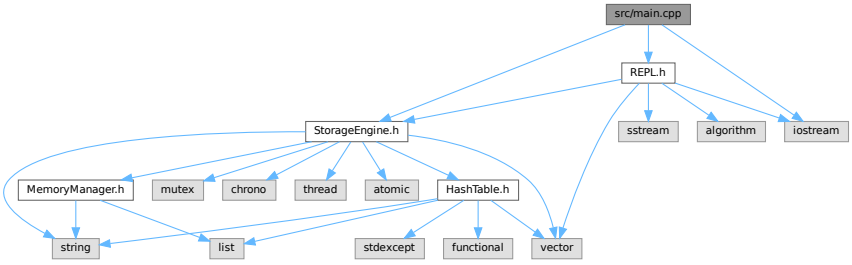
```
00190     {
00191         size_t index = hash(key);
00192         Node *prev = nullptr;
00193         Node *current = table[index];
00194 
00195         while (current)
00196         {
00197             if (current->key == key)
00198             {
00199                 if (prev)
00200                 {
00201                     prev->next = current->next;
00202                 }
00203                 else
00204                 {
00205                     table[index] = current->next;
00206                 }
00207 
00208                 delete current;
00209                 size--;
00210 
00211                 if (capacity > 8 && size < 0.2 * capacity)
00212                 {
00213                     resize(capacity / 2);
00214                 }
00215                 return true;
00216             }
00217 
00218             prev = current;
00219             current = current->next;
00220         }
00221         return false;
00222     }
00223 
00228     void clear()
00229     {
00230         for (size_t i = 0; i < capacity; ++i)
00231         {
00232             Node *current = table[i];
00233             while (current)
00234             {
00235                 Node *next = current->next;
00236                 delete current;
00237                 current = next;
00238             }
00239             table[i] = nullptr;
00240         }
00241         size = 0;
00242     }
00243 
00244     // Disable copy operations
00245     HashTable(const HashTable &) = delete;
00246     HashTable &operator=(const HashTable &) = delete;
00247 };
```

4.3 src/main.cpp File Reference

Entry point for BLINK DB Storage Engine (Part 1)

```
#include "StorageEngine.h"
#include "REPL.h"
#include <iostream>
```

Include dependency graph for main.cpp:



Functions

- int main ()

  Main function for BLINK DB Storage Engine.

4.3.1 Detailed Description

Entry point for BLINK DB Storage Engine (Part 1)

Implements the REPL interface for interacting with the key-value store. Demonstrates the core functionality of SET, GET, and DEL operations.

4.3.2 Function Documentation

4.3.2.1 main()

```
int main ( )
```

Main function for BLINK DB Storage Engine.

Initializes the storage engine and REPL interface, then enters the command processing loop. Handles user input for database operations until explicit exit command.

Returns

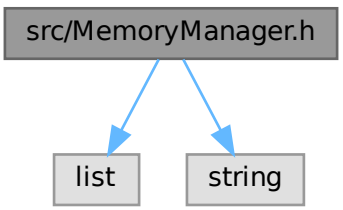
int Exit status (0 for normal termination)

4.4 src/MemoryManager.h File Reference

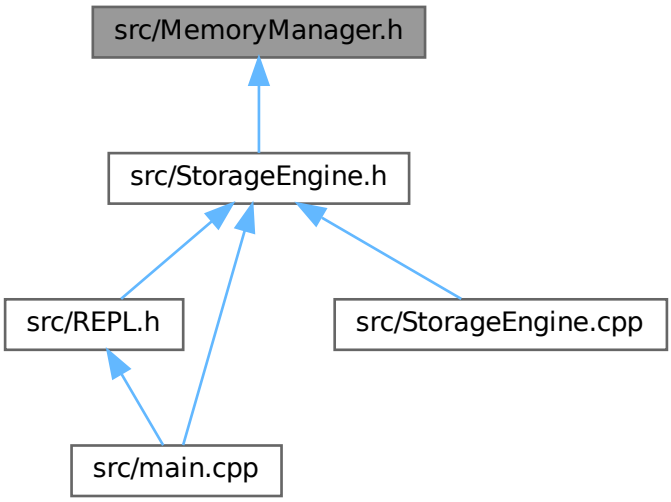
LRU eviction policy manager for BLINK DB storage engine.

```
#include <list>
#include <string>
```

Include dependency graph for MemoryManager.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MemoryManager

  Manages Least Recently Used (LRU) eviction policy.

4.4.1 Detailed Description

LRU eviction policy manager for BLINK DB storage engine.

4.5 MemoryManager.h

Go to the documentation of this file.

```
00001
00006  #pragma once
00007  #include <list>
00008  #include <string>
00009
00017  class MemoryManager {
00018  private:
00019      std::list<std::string> lru_queue;
00020
00021  public:
00029      void update_lru(const std::string& key) {
00030          lru_queue.remove(key);
00031          lru_queue.push_front(key);
00032      }
00033
00041      std::string evict_lru() {
00042          if (!lru_queue.empty()) {
00043              std::string key = lru_queue.back();
00044              lru_queue.pop_back();
00045              return key;
00046          }
00047          return "";
```

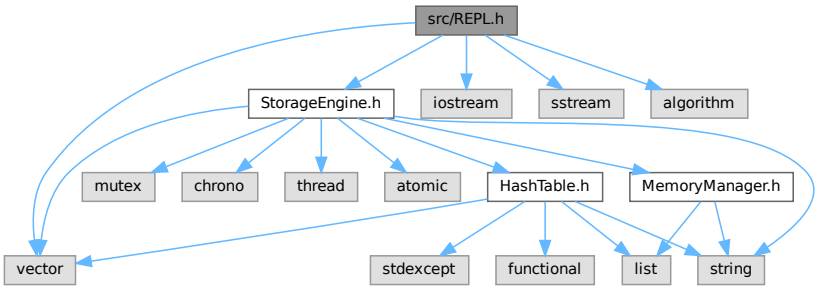
```
00048       }
00049
00057       void evict_lru(const std::string& key) {
00058           lru_queue.remove(key);
00059       }
00060   };
00061
```

4.6 src/REPL.h File Reference

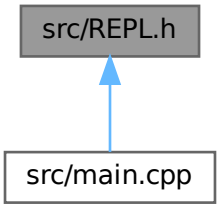
Read-Eval-Print Loop interface for BLINK DB (Part 1)

```
#include "StorageEngine.h"
#include <iostream>
#include <sstream>
#include <vector>
#include <algorithm>
```

Include dependency graph for REPL.h:



This graph shows which files directly or indirectly include this file:



Classes

- class REPL

  Command processor for BLINK DB storage engine.

4.6.1 Detailed Description

Read-Eval-Print Loop interface for BLINK DB (Part 1)

Implements the command-line interface for interacting with the storage engine. Separates user input handling from core database logic as per project requirements.

4.7 REPL.h

Go to the documentation of this file.

```
00001
00009  #pragma once
00010  #include "StorageEngine.h"
00011  #include <iostream>
00012  #include <sstream>
00013  #include <vector>
00014  #include <algorithm>
00015
00027  class REPL {
00028      StorageEngine& engine;
00029
00040      std::vector<std::string> tokenize(const std::string& input) {
00041          std::vector<std::string> tokens;
00042          std::istringstream ss(input);
00043          std::string token;
00044          bool in_quotes = false;
00045          char quote_char = 0;
00046
00047          while(ss » std::ws) {
00048              if(ss.peek() == '"' || ss.peek() == '\'') {
00049                  in_quotes = !in_quotes;
00050                  quote_char = ss.get();
00051                  std::getline(ss, token, quote_char);
00052                  if(!token.empty()) {
00053                      tokens.push_back(token);
00054                  }
00055                  continue;
00056              }
00057
00058              if(in_quotes) {
00059                  std::getline(ss, token, quote_char);
00060                  in_quotes = false;
00061                  if(!token.empty()) tokens.push_back(token);
00062              }
00063              else {
00064                  ss » token;
00065                  if(!token.empty()) tokens.push_back(token);
00066              }
00067          }
00068          return tokens;
00069      }
00070
00071  public:
00076      REPL(StorageEngine& engine) : engine(engine) {}
00077
00091      void process_command(const std::string& input) {
00092          auto tokens = tokenize(input);
00093          if(tokens.empty()) return;
00094
00095          try {
00096              std::transform(tokens[0].begin(), tokens[0].end(), tokens[0].begin(), ::toupper);
00097
00098              if(tokens[0] == "SET" && tokens.size() >= 3) {
00099                  // Handle TTL if specified
00100                  std::chrono::seconds ttl = std::chrono::seconds::max();
00101                  if(tokens.size() >= 5 && tokens[3] == "EX") {
00102                      ttl = std::chrono::seconds(std::stoi(tokens[4]));
00103                  }
00104
00105                  engine.set(tokens[1], tokens[2], ttl);
00106                 //  std::cout « "OK" « std::endl;
00107              }
00108              else if(tokens[0] == "GET" && tokens.size() >= 2) {
00109                  auto value = engine.get(tokens[1]);
00110                  std::cout « (value.empty() ? "NULL" : value) « std::endl;
00111              }
00112              else if(tokens[0] == "DEL" && tokens.size() >= 2) {
00113                  bool deleted = engine.del(tokens[1]);
```

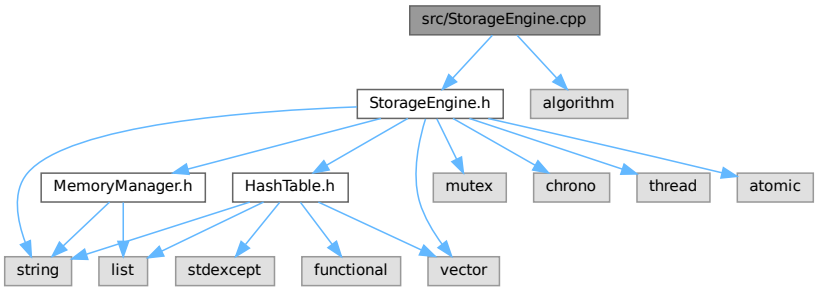
```
00114                 if(!deleted) std::cout « "Does not exist.\n";
00115             }
00116             else {
00117                 std::cout « "ERROR: Invalid command format" « std::endl;
00118             }
00119         }
00120         catch(const std::invalid_argument& e) {
00121             std::cout « "ERROR: Invalid numeric argument" « std::endl;
00122         }
00123         catch(const std::exception& e) {
00124             std::cout « "ERROR: " « e.what() « std::endl;
00125         }
00126     }
00127 };
00128
```

4.8 src/StorageEngine.cpp File Reference

Core implementation of BLINK DB storage engine.

```
#include "StorageEngine.h"
#include <algorithm>
```

Include dependency graph for StorageEngine.cpp:



4.8.1 Detailed Description

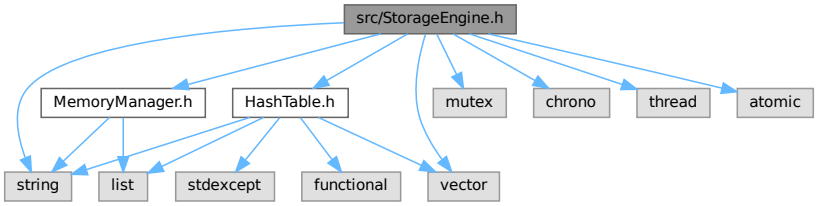
Core implementation of BLINK DB storage engine.

4.9 src/StorageEngine.h File Reference

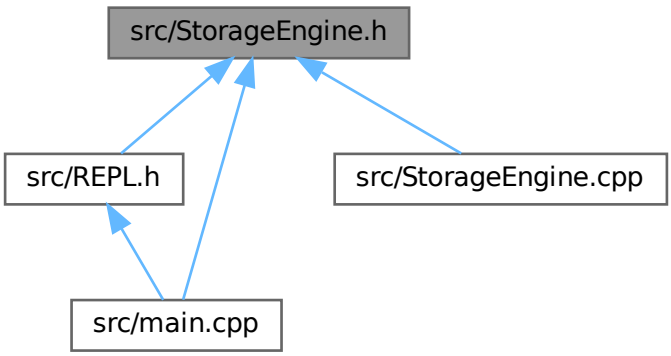
Core header for BLINK DB storage engine implementation.

```
#include <string>
#include <mutex>
#include <chrono>
#include <thread>
#include <atomic>
#include <vector>
#include "HashTable.h"
```

`#include "MemoryManager.h"`
Include dependency graph for StorageEngine.h:



This graph shows which files directly or indirectly include this file:



Classes

- class StorageEngine

  In-memory key-value store with LRU eviction and TTL expiration.

4.9.1 Detailed Description

Core header for BLINK DB storage engine implementation.

4.10 StorageEngine.h

Go to the documentation of this file.

```
00001
00006  #pragma once
00007  #include <string>
00008  #include <mutex>
```

```
00009  #include <chrono>
00010  #include <thread>
00011  #include <atomic>
00012  #include <vector>
00013  #include "HashTable.h"
00014  #include "MemoryManager.h"
00015
00026  class StorageEngine {
00027  private:
00032      struct Entry {
00033          std::string value;
00034          std::chrono::seconds ttl;
00035          std::chrono::system_clock::time_point last_accessed;
00036      };
00037
00038      HashTable<std::string, Entry> store;
00039      MemoryManager mem_manager;
00040      size_t current_memory = 0;
00041      size_t max_memory;
00042      std::mutex mtx;
00043      std::thread eviction_thread;
00044      std::atomic<bool> running{true};
00045
00050      void start_eviction_daemon() {
00051          eviction_thread = std::thread([this] {
00052              while (running) {
00053                  std::this_thread::sleep_for(std::chrono::seconds(1));
00054                  evict_expired();
00055              }
00056          });
00057      }
00058
00059  public:
00064      explicit StorageEngine(size_t max_memory = 1024 * 1024 * 1024);
00065
00070      ~StorageEngine();
00071
00080      void set(const std::string& key, const std::string& value,
00081               std::chrono::seconds ttl = std::chrono::seconds::max());
00082
00091      std::string get(const std::string& key);
00092
00100      bool del(const std::string& key);
00101
00102  private:
00107      void enforce_memory_limits();
00108
00114      void evict_expired();
00115  };
00116
```

Index