

Project 4 MapReduce

1

Generated by Doxygen 1.8.17

1 Namespace Index	1
1.1 Namespace List	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Data Structure Index	5
3.1 Data Structures	5
4 File Index	7
4.1 File List	7
5 Namespace Documentation	9
5.1 anonymous_namespace{mr_task_factory.cc} Namespace Reference	9
6 Data Structure Documentation	11
6.1 AsyncClientCall Class Reference	11
6.1.1 Detailed Description	12
6.1.2 Constructor & Destructor Documentation	12
6.1.2.1 ~AsyncClientCall()	12
6.1.3 Field Documentation	12
6.1.3.1 context	12
6.1.3.2 is_map_job	12
6.1.3.3 status	13
6.1.3.4 worker_ip_addr	13
6.2 BaseHandler Class Reference	13
6.2.1 Detailed Description	14
6.2.2 Member Enumeration Documentation	14
6.2.2.1 CallStatus	14
6.2.3 Constructor & Destructor Documentation	14
6.2.3.1 BaseHandler()	15
6.2.3.2 ~BaseHandler()	15
6.2.4 Member Function Documentation	15
6.2.4.1 Proceed()	15
6.2.5 Field Documentation	16
6.2.5.1 ctx_	16
6.2.5.2 s_queue	16
6.2.5.3 service	16
6.2.5.4 status_	16
6.2.5.5 worker_address	17
6.3 BaseMapperInternal Struct Reference	17
6.3.1 Detailed Description	17
6.3.2 Constructor & Destructor Documentation	17
6.3.2.1 BaseMapperInternal()	18

6.3.3 Member Function Documentation	18
6.3.3.1 emit()	18
6.3.3.2 final_flush()	19
6.3.3.3 internal_file_mapping()	19
6.3.4 Field Documentation	20
6.3.4.1 intermediate_file_list	20
6.3.4.2 kv_pair_vector	20
6.4 BaseReducerInternal Struct Reference	21
6.4.1 Detailed Description	21
6.4.2 Constructor & Destructor Documentation	21
6.4.2.1 BaseReducerInternal()	21
6.4.3 Member Function Documentation	22
6.4.3.1 emit()	22
6.4.4 Field Documentation	22
6.4.4.1 file_name	22
6.5 FileShard Struct Reference	23
6.5.1 Detailed Description	23
6.5.2 Field Documentation	23
6.5.2.1 shard_id	24
6.5.2.2 split_file_list	24
6.6 heartbeat_payload Struct Reference	24
6.6.1 Detailed Description	25
6.6.2 Field Documentation	25
6.6.2.1 id	25
6.6.2.2 timestamp	25
6.6.2.3 workerStatus	25
6.7 HeartbeatCall Class Reference	26
6.7.1 Detailed Description	26
6.7.2 Field Documentation	26
6.7.2.1 heartbeat_payload_reader	27
6.7.2.2 result	27
6.8 HeartbeatHandler Class Reference	27
6.8.1 Detailed Description	28
6.8.2 Constructor & Destructor Documentation	28
6.8.2.1 HeartbeatHandler()	28
6.8.3 Member Function Documentation	29
6.8.3.1 handle_heartbeat_job()	29
6.8.3.2 Proceed()	30
6.8.4 Field Documentation	31
6.8.4.1 h_writer	31
6.8.4.2 request	31
6.8.4.3 response	31

6.9 MapCall Class Reference	32
6.9.1 Detailed Description	32
6.9.2 Field Documentation	32
6.9.2.1 map_response_reader	33
6.9.2.2 result	33
6.10 MapperHandler Class Reference	33
6.10.1 Detailed Description	34
6.10.2 Constructor & Destructor Documentation	34
6.10.2.1 MapperHandler()	34
6.10.3 Member Function Documentation	35
6.10.3.1 convert_grpc_spec()	35
6.10.3.2 get_basemapper_internal()	36
6.10.3.3 handle_mapper_job()	37
6.10.3.4 Proceed()	38
6.10.4 Field Documentation	39
6.10.4.1 m_writer	39
6.10.4.2 mapRequest	39
6.10.4.3 mapResponse	40
6.11 MapReduceImpl Class Reference	40
6.11.1 Detailed Description	41
6.11.2 Member Function Documentation	41
6.11.2.1 create_shards()	41
6.11.2.2 read_and_validate_spec()	42
6.11.2.3 run()	42
6.11.2.4 run_master()	43
6.11.3 Field Documentation	44
6.11.3.1 file_shards_	44
6.11.3.2 mr_spec_	44
6.12 MapReduceSpec Struct Reference	45
6.12.1 Detailed Description	45
6.12.2 Field Documentation	45
6.12.2.1 input_files	46
6.12.2.2 map_kb	46
6.12.2.3 output_directory	46
6.12.2.4 output_files	46
6.12.2.5 user	46
6.12.2.6 worker_count	47
6.12.2.7 worker_endpoints	47
6.13 Master Class Reference	47
6.13.1 Detailed Description	48
6.13.2 Constructor & Destructor Documentation	48
6.13.2.1 Master()	48

6.13.2.2 ~Master()	49
6.13.3 Member Function Documentation	49
6.13.3.1 assign_files_to_reducer()	49
6.13.3.2 async_map()	50
6.13.3.3 async_reducer()	52
6.13.3.4 cleanup_files()	53
6.13.3.5 find_worker_by_name()	54
6.13.3.6 find_worker_by_status()	55
6.13.3.7 handler_dead_worker()	56
6.13.3.8 heartbeat()	57
6.13.3.9 run()	58
6.13.4 Field Documentation	61
6.13.4.1 assigned_partition	61
6.13.4.2 assigned_shards	61
6.13.4.3 cleanup_mutex	62
6.13.4.4 completion_count	62
6.13.4.5 condition_cleanup_mutex	62
6.13.4.6 condition_heartbeat	62
6.13.4.7 condition_ops_mutex	62
6.13.4.8 condition_worker_queue_mutex	63
6.13.4.9 cq_	63
6.13.4.10 dummy	63
6.13.4.11 file_shards	63
6.13.4.12 heartbeat_mutex	63
6.13.4.13 init_heartbeat	64
6.13.4.14 intermidateFiles	64
6.13.4.15 missing_output_files	64
6.13.4.16 missing_shards	64
6.13.4.17 mr_spec	64
6.13.4.18 ops_completed	65
6.13.4.19 ops_mutex	65
6.13.4.20 OutputFiles	65
6.13.4.21 server_state	65
6.13.4.22 worker_queue_mutex	65
6.13.4.23 workers	66
6.14 ReduceCall Class Reference	66
6.14.1 Detailed Description	67
6.14.2 Field Documentation	67
6.14.2.1 reducer_response_reader	67
6.14.2.2 result	67
6.15 ReducerHandler Class Reference	67
6.15.1 Detailed Description	68

6.15.2 Constructor & Destructor Documentation	68
6.15.2.1 ReducerHandler()	68
6.15.3 Member Function Documentation	69
6.15.3.1 get_basereducer_internal()	69
6.15.3.2 handle_reducer_job()	70
6.15.3.3 Proceed()	71
6.15.4 Field Documentation	72
6.15.4.1 r_writer	72
6.15.4.2 ReduceRequest	73
6.15.4.3 ReduceResponse	73
6.16 splitFile Struct Reference	73
6.16.1 Detailed Description	74
6.16.2 Field Documentation	74
6.16.2.1 filename	74
6.16.2.2 offsets	74
6.17 anonymous_namespace{mr_task_factory.cc}::TaskFactory Class Reference	74
6.17.1 Detailed Description	75
6.17.2 Constructor & Destructor Documentation	75
6.17.2.1 TaskFactory()	75
6.17.3 Member Function Documentation	75
6.17.3.1 get_mapper()	75
6.17.3.2 get_reducer()	76
6.17.3.3 instance()	76
6.17.4 Field Documentation	76
6.17.4.1 mappers_	76
6.17.4.2 reducers_	76
6.18 Worker Class Reference	77
6.18.1 Detailed Description	78
6.18.2 Constructor & Destructor Documentation	78
6.18.2.1 Worker()	78
6.18.2.2 ~Worker()	78
6.18.3 Member Function Documentation	78
6.18.3.1 get_basemapper_internal()	79
6.18.3.2 get_basereducer_internal()	79
6.18.3.3 heartbeat_handler()	79
6.18.3.4 run()	80
6.18.4 Field Documentation	81
6.18.4.1 builder	81
6.18.4.2 clean_exit	81
6.18.4.3 heartbeat_queue	81
6.18.4.4 mapreduce_service	82
6.18.4.5 server	82

6.18.4.6 work_queue	82
6.18.4.7 worker_uuid	82
6.19 worker Struct Reference	82
6.19.1 Detailed Description	83
6.19.2 Field Documentation	83
6.19.2.1 client	83
6.19.2.2 current_output	83
6.19.2.3 current_shard	83
6.19.2.4 dead_handled	84
6.19.2.5 output_reducer_location_map	84
6.19.2.6 worker_address	84
6.19.2.7 workerStatus	84
6.19.2.8 workerType	84
6.20 WorkerClient Class Reference	85
6.20.1 Detailed Description	85
6.20.2 Constructor & Destructor Documentation	86
6.20.2.1 WorkerClient()	86
6.20.2.2 ~WorkerClient()	86
6.20.3 Member Function Documentation	86
6.20.3.1 convert_grpc_spec()	86
6.20.3.2 recv_heartbeat()	87
6.20.3.3 schedule_mapper_jobs()	88
6.20.3.4 schedule_reduce_job()	88
6.20.3.5 send_heartbeat()	89
6.20.4 Field Documentation	90
6.20.4.1 heartbeat_queue	90
6.20.4.2 queue	90
6.20.4.3 stub	90
6.20.4.4 worker_address	90
7 File Documentation	91
7.1 src/CMakeLists.txt File Reference	91
7.1.1 Function Documentation	91
7.1.1.1 add_dependencies()	91
7.1.1.2 cmake_minimum_required()	91
7.2 src/file_shard.h File Reference	92
7.2.1 Macro Definition Documentation	93
7.2.1.1 KB	93
7.2.1.2 TEMP_DIR	93
7.2.2 Function Documentation	93
7.2.2.1 approx_split()	93
7.2.2.2 get_filesize()	94

7.2.2.3 shard_files()	95
7.3 src/mapreduce.cc File Reference	96
7.4 src/mapreduce_impl.cc File Reference	97
7.5 src/mapreduce_impl.h File Reference	97
7.6 src/mapreduce_spec.h File Reference	98
7.6.1 Function Documentation	99
7.6.1.1 read_mr_spec_from_config_file()	99
7.6.1.2 splitString()	101
7.6.1.3 validate_mr_spec()	102
7.7 src/master.h File Reference	103
7.7.1 Macro Definition Documentation	104
7.7.1.1 ALIVE	104
7.7.1.2 TIMEOUT	104
7.7.2 Enumeration Type Documentation	104
7.7.2.1 WORKER_STATUS	104
7.7.2.2 WORKER_TYPE	105
7.8 src/mr_task_factory.cc File Reference	105
7.8.1 Function Documentation	106
7.8.1.1 get_mapper_from_task_factory()	106
7.8.1.2 get_reducer_from_task_factory()	107
7.8.1.3 register_tasks()	107
7.9 src/mr_tasks.h File Reference	107
7.9.1 Macro Definition Documentation	108
7.9.1.1 DEBUG	108
7.9.1.2 DELIMITER	108
7.9.1.3 devnull	109
7.9.1.4 MAX_KV_PAIR_SIZE	109
7.10 src/Readme.txt File Reference	109
7.11 src/run_worker.cc File Reference	109
7.11.1 Function Documentation	109
7.11.1.1 main()	110
7.12 src/worker.h File Reference	110
7.12.1 Function Documentation	111
7.12.1.1 get_mapper_from_task_factory()	111
7.12.1.2 get_reducer_from_task_factory()	112

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

anonymous_namespace{mr_task_factory.cc}	9
---	---

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AsyncClientCall	11
HeartbeatCall	26
MapCall	32
ReduceCall	66
BaseHandler	13
HeartbeatHandler	27
MapperHandler	33
ReducerHandler	67
BaseMapperInternal	17
BaseReducerInternal	21
FileShard	23
heartbeat_payload	24
MapReduceImpl	40
MapReduceSpec	45
Master	47
splitFile	73
anonymous_namespace{mr_task_factory.cc}::TaskFactory	74
Worker	77
worker	82
WorkerClient	85

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

AsyncClientCall	11
BaseHandler	13
BaseMapperInternal	17
BaseReducerInternal	21
FileShard	23
heartbeat_payload	24
HeartbeatCall	26
HeartbeatHandler	27
MapCall	32
MapperHandler	33
MapReduceImpl	40
MapReduceSpec	45
Master	47
ReduceCall	66
ReducerHandler	67
splitFile	73
anonymous_namespace{mr_task_factory.cc}::TaskFactory	74
Worker	77
worker	82
WorkerClient	85

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

src/ file_shard.h	92
src/ mapreduce.cc	96
src/ mapreduce_impl.cc	97
src/ mapreduce_impl.h	97
src/ mapreduce_spec.h	98
src/ master.h	103
src/ mr_task_factory.cc	105
src/ mr_tasks.h	107
src/ run_worker.cc	109
src/ worker.h	110

Chapter 5

Namespace Documentation

5.1 anonymous_namespace{mr_task_factory.cc} Namespace Reference

Data Structures

- class [TaskFactory](#)

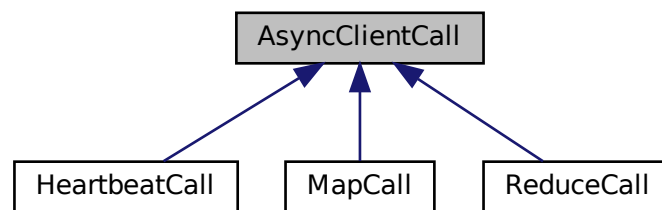
Chapter 6

Data Structure Documentation

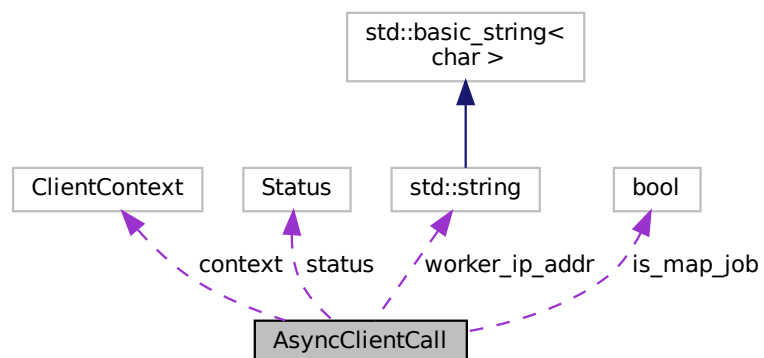
6.1 AsyncClientCall Class Reference

```
#include <master.h>
```

Inheritance diagram for AsyncClientCall:



Collaboration diagram for AsyncClientCall:



Public Member Functions

- virtual `~AsyncClientCall()`=default

Data Fields

- bool `is_map_job` = true
- `grpc::ClientContext` `context`
- `grpc::Status` `status`
- `std::string` `worker_ip_addr`

6.1.1 Detailed Description

Base Class to handle all Async Response.

Definition at line 47 of file master.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 `~AsyncClientCall()`

```
virtual AsyncClientCall::~AsyncClientCall ( ) [virtual], [default]
```

6.1.3 Field Documentation

6.1.3.1 `context`

```
grpc::ClientContext AsyncClientCall::context
```

Definition at line 51 of file master.h.

6.1.3.2 `is_map_job`

```
bool AsyncClientCall::is_map_job = true
```

Definition at line 50 of file master.h.

6.1.3.3 status

```
grpc::Status AsyncClientCall::status
```

Definition at line 52 of file master.h.

6.1.3.4 worker_ip_addr

```
std::string AsyncClientCall::worker_ip_addr
```

Definition at line 53 of file master.h.

Referenced by WorkerClient::recv_heartbeat(), WorkerClient::schedule_mapper_jobs(), WorkerClient::schedule_reduce_job(), and WorkerClient::send_heartbeat().

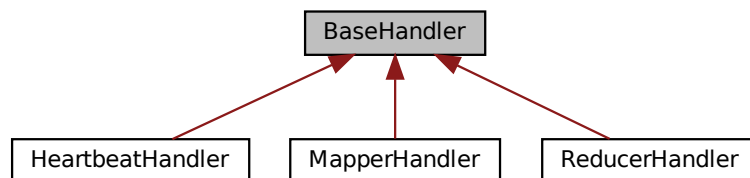
The documentation for this class was generated from the following file:

- [src/master.h](#)

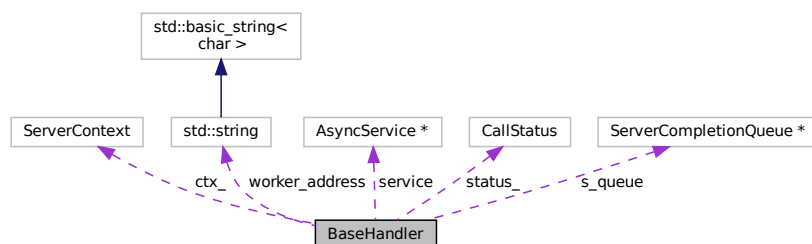
6.2 BaseHandler Class Reference

```
#include <worker.h>
```

Inheritance diagram for BaseHandler:



Collaboration diagram for BaseHandler:



Public Member Functions

- [BaseHandler](#) (masterworker::Map_Reduce::AsyncService *[service](#), grpc::ServerCompletionQueue *[queue](#), std::string [worker_address](#))
- virtual void [Proceed](#) ()
- [~BaseHandler](#) ()=default

Protected Types

- enum [CallStatus](#) { [CREATE](#), [PROCESS](#), [FINISH](#) }

Protected Attributes

- masterworker::Map_Reduce::AsyncService * [service](#)
- grpc::ServerCompletionQueue * [s_queue](#)
- std::string [worker_address](#)
- grpc::ServerContext [ctx_](#)
- [CallStatus](#) [status_](#)

6.2.1 Detailed Description

Base Class for Three task , map , reduce and heartbeat

Definition at line 25 of file worker.h.

6.2.2 Member Enumeration Documentation

6.2.2.1 CallStatus

```
enum BaseHandler::CallStatus [protected]
```

Enumerator

CREATE	
PROCESS	
FINISH	

Definition at line 52 of file worker.h.

```
53     {
54         CREATE,
55         PROCESS,
56         FINISH
57     };
```

6.2.3 Constructor & Destructor Documentation

6.2.3.1 BaseHandler()

```
BaseHandler::BaseHandler (
    masterworker::Map_Reduce::AsyncService * service,
    grpc::ServerCompletionQueue * queue,
    std::string worker_address ) [inline]
```

Definition at line 28 of file worker.h.

```
32      : service(service)
33      , s_queue(queue)
34      , worker_address(std::move(worker_address))
35      , status_(CREATE)
36      {
37      }
38      Proceed();
```

References `Proceed()`.

Here is the call graph for this function:



6.2.3.2 ~BaseHandler()

```
BaseHandler::~~BaseHandler ( ) [default]
```

6.2.4 Member Function Documentation

6.2.4.1 Proceed()

```
virtual void BaseHandler::Proceed ( ) [inline], [virtual]
```

Reimplemented in [HeartbeatHandler](#), [ReducerHandler](#), and [MapperHandler](#).

Definition at line 40 of file worker.h.

```
41      {
42      }
```

Referenced by `BaseHandler()`.

Here is the caller graph for this function:



6.2.5 Field Documentation

6.2.5.1 ctx_

```
grpc::ServerContext BaseHandler::ctx_ [protected]
```

Definition at line 51 of file worker.h.

Referenced by MapperHandler::Proceed(), ReducerHandler::Proceed(), and HeartbeatHandler::Proceed().

6.2.5.2 s_queue

```
grpc::ServerCompletionQueue* BaseHandler::s_queue [protected]
```

Definition at line 48 of file worker.h.

Referenced by MapperHandler::Proceed(), ReducerHandler::Proceed(), and HeartbeatHandler::Proceed().

6.2.5.3 service

```
masterworker::Map_Reduce::AsyncService* BaseHandler::service [protected]
```

Definition at line 47 of file worker.h.

Referenced by MapperHandler::Proceed(), ReducerHandler::Proceed(), and HeartbeatHandler::Proceed().

6.2.5.4 status_

```
CallStatus BaseHandler::status_ [protected]
```

Definition at line 58 of file worker.h.

Referenced by MapperHandler::Proceed(), ReducerHandler::Proceed(), and HeartbeatHandler::Proceed().

6.2.5.5 worker_address

```
std::string BaseHandler::worker_address [protected]
```

Definition at line 49 of file worker.h.

Referenced by MapperHandler::handle_mapper_job(), MapperHandler::Proceed(), ReducerHandler::Proceed(), and HeartbeatHandler::Proceed().

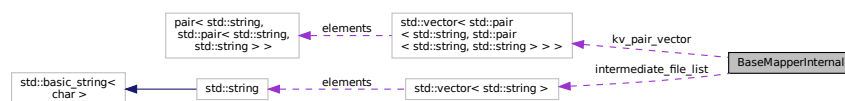
The documentation for this class was generated from the following file:

- src/[worker.h](#)

6.3 BaseMapperInternal Struct Reference

```
#include <mr_tasks.h>
```

Collaboration diagram for BaseMapperInternal:



Public Member Functions

- [BaseMapperInternal](#) ()
- void [emit](#) (const std::string &key, const std::string &val)
- std::string [internal_file_mapping](#) (std::string key)
- void [final_flush](#) ()

Data Fields

- std::vector< std::pair< std::string, std::pair< std::string, std::string > > > [kv_pair_vector](#)
- std::vector< std::string > [intermediate_file_list](#)

6.3.1 Detailed Description

Definition at line 16 of file mr_tasks.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 BaseMapperInternal()

```
BaseMapperInternal::BaseMapperInternal ( ) [inline]
```

Constructor not required as no private variable require initialization

Definition at line 43 of file mr_tasks.h.

```
44 {
45 }
```

6.3.3 Member Function Documentation

6.3.3.1 emit()

```
void BaseMapperInternal::emit (
    const std::string & key,
    const std::string & val ) [inline]
```

Flush Key Value pair to required intermediate file. We use Caching of MAX_KV_PAIR_SIZE (2048)

Parameters

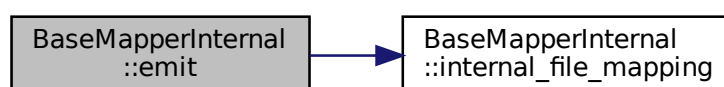
<i>key</i>	
<i>val</i>	

Definition at line 67 of file mr_tasks.h.

```
68 {
69 #if DEBUG > 1
70 //     std::cout << BaseMapperInternal::kv_pair_vector.size() << "Dummy emit by BaseMapperInternal: " << key
71 //     << DELIMITER
72 //     << val << std::endl;
73 #endif
74 if (BaseMapperInternal::kv_pair_vector.size() > MAX_KV_PAIR_SIZE)
75 {
76     for (const auto& a : BaseMapperInternal::kv_pair_vector)
77     {
78         std::ofstream f(a.first, std::ofstream::out | std::ofstream::app);
79         f << a.second.first << DELIMITER << a.second.second << std::endl;
80     }
81     BaseMapperInternal::kv_pair_vector.clear();
82     BaseMapperInternal::kv_pair_vector.push_back({BaseMapperInternal::internal_file_mapping(key), {key,
83         val}});
84 }
```

References DELIMITER, internal_file_mapping(), kv_pair_vector, and MAX_KV_PAIR_SIZE.

Here is the call graph for this function:



6.3.3.2 final_flush()

```
void BaseMapperInternal::final_flush ( ) [inline]
```

Final flush to intermediate file for given map operation.

Definition at line 87 of file mr_tasks.h.

```
88 {
89     for (const auto& a : BaseMapperInternal::kv_pair_vector)
90     {
91         std::ofstream f(a.first, std::ofstream::out | std::ofstream::app);
92         f << a.second.first << DELIMITER << a.second.second << std::endl;
93         f.close();
94     }
95     BaseMapperInternal::kv_pair_vector.clear();
96 }
```

References DELIMITER, and kv_pair_vector.

6.3.3.3 internal_file_mapping()

```
std::string BaseMapperInternal::internal_file_mapping (
    std::string key ) [inline]
```

Find intermediate file based on given key by creating hash of key and taking modulo. Taking hash allows normal distribution of keys for unique keys.

Parameters

key	
-----	--

Returns

file location for given key.

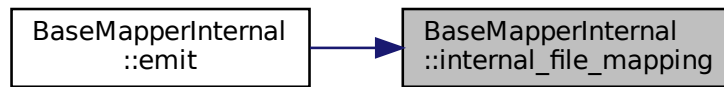
Definition at line 53 of file mr_tasks.h.

```
54 {
55     std::hash<std::string> h;
56     if (BaseMapperInternal::intermediate_file_list.empty())
57         return devnull;
58     auto file_location = h(key) % BaseMapperInternal::intermediate_file_list.size();
59     return BaseMapperInternal::intermediate_file_list[file_location];
60 }
```

References devnull, and intermediate_file_list.

Referenced by emit().

Here is the caller graph for this function:



6.3.4 Field Documentation

6.3.4.1 intermediate_file_list

```
std::vector<std::string> BaseMapperInternal::intermediate_file_list
```

Definition at line 32 of file mr_tasks.h.

Referenced by `internal_file_mapping()`.

6.3.4.2 kv_pair_vector

```
std::vector<std::pair<std::string, std::pair<std::string, std::string> > > BaseMapperInternal::kv_pair_vector
```

Storage vector for key value pairs

Definition at line 30 of file mr_tasks.h.

Referenced by `emit()`, and `final_flush()`.

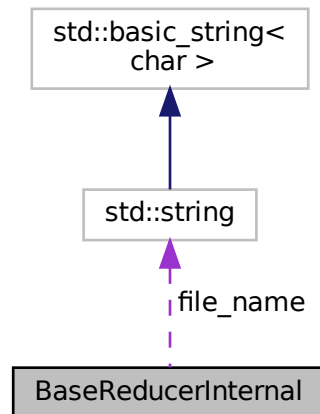
The documentation for this struct was generated from the following file:

- [src/mr_tasks.h](#)

6.4 BaseReducerInternal Struct Reference

```
#include <mr_tasks.h>
```

Collaboration diagram for BaseReducerInternal:



Public Member Functions

- [BaseReducerInternal](#) ()
- void [emit](#) (const std::string &key, const std::string &val)

Data Fields

- std::string [file_name](#)

6.4.1 Detailed Description

Definition at line 102 of file `mr_tasks.h`.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 BaseReducerInternal()

```
BaseReducerInternal::BaseReducerInternal ( ) [inline]
```

Constructor not required as no private variable require initialization

Definition at line 119 of file `mr_tasks.h`.

```
120 {  
121 }
```

6.4.3 Member Function Documentation

6.4.3.1 emit()

```
void BaseReducerInternal::emit (
    const std::string & key,
    const std::string & val ) [inline]
```

Emit given key value pair to output file this->file_name.

Parameters

<i>key</i>	
<i>val</i>	

Definition at line 129 of file mr_tasks.h.

```
130 {
131     #if DEBUG > 1
132         std::cout << "Dummy emit by BaseReducerInternal: " << key << ", " << val << std::endl;
133     #endif
134     std::ofstream f(file_name, std::ofstream::out | std::ofstream::app);
135     f << key << " " << val << std::endl;
136     f.close();
137 }
```

References file_name.

6.4.4 Field Documentation

6.4.4.1 file_name

```
std::string BaseReducerInternal::file_name
```

Definition at line 113 of file mr_tasks.h.

Referenced by emit().

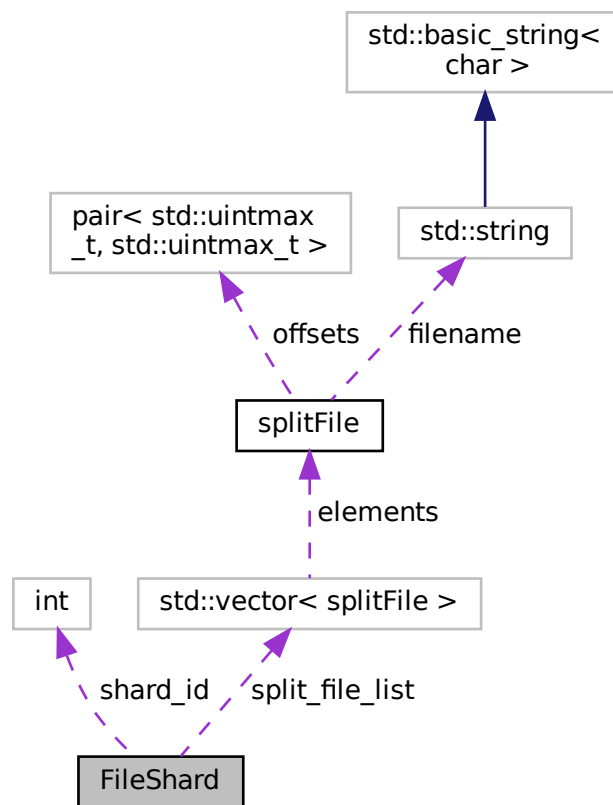
The documentation for this struct was generated from the following file:

- [src/mr_tasks.h](#)

6.5 FileShard Struct Reference

```
#include <file_shard.h>
```

Collaboration diagram for FileShard:



Data Fields

- `int shard_id = -1`
- `std::vector< splitFile > split_file_list`

6.5.1 Detailed Description

Definition at line 33 of file `file_shard.h`.

6.5.2 Field Documentation

6.5.2.1 shard_id

```
int FileShard::shard_id = -1
```

Definition at line 35 of file file_shard.h.

Referenced by MapperHandler::convert_grpc_spec(), WorkerClient::convert_grpc_spec(), and shard_files().

6.5.2.2 split_file_list

```
std::vector<splitFile> FileShard::split_file_list
```

Definition at line 36 of file file_shard.h.

Referenced by WorkerClient::convert_grpc_spec(), MapperHandler::handle_mapper_job(), and shard_files().

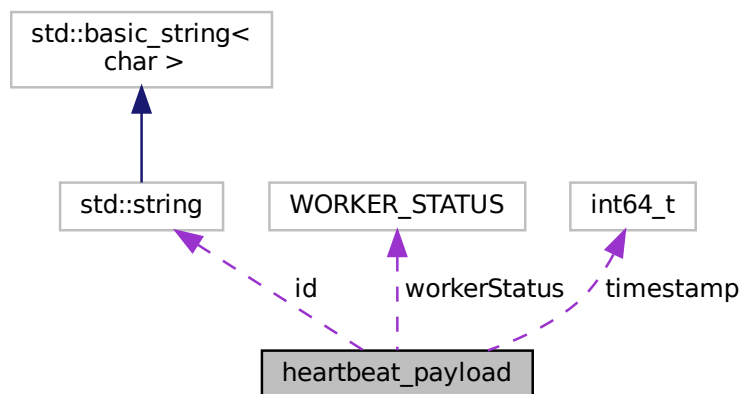
The documentation for this struct was generated from the following file:

- src/file_shard.h

6.6 heartbeat_payload Struct Reference

```
#include <master.h>
```

Collaboration diagram for heartbeat_payload:



Data Fields

- std::string [id](#)
- std::int64_t [timestamp](#)
- [WORKER_STATUS](#) [workerStatus](#)

6.6.1 Detailed Description

Definition at line 37 of file master.h.

6.6.2 Field Documentation

6.6.2.1 id

```
std::string heartbeat_payload::id
```

Definition at line 39 of file master.h.

Referenced by Master::heartbeat().

6.6.2.2 timestamp

```
std::int64_t heartbeat_payload::timestamp
```

Definition at line 40 of file master.h.

6.6.2.3 workerStatus

```
WORKER_STATUS heartbeat_payload::workerStatus
```

Definition at line 41 of file master.h.

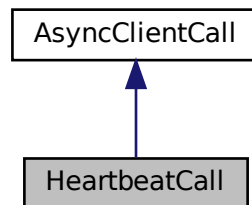
The documentation for this struct was generated from the following file:

- [src/master.h](#)

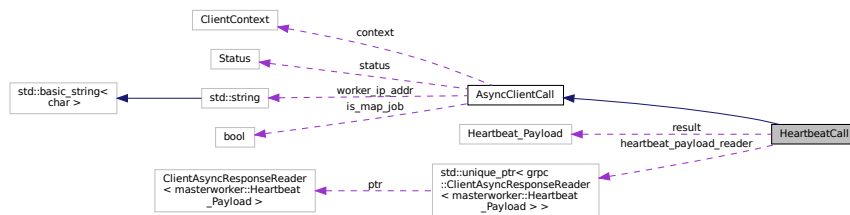
6.7 HeartbeatCall Class Reference

```
#include <master.h>
```

Inheritance diagram for HeartbeatCall:



Collaboration diagram for HeartbeatCall:



Data Fields

- `masterworker::Heartbeat_Payload` [result](#)
- `std::unique_ptr< grpc::ClientAsyncResponseReader< masterworker::Heartbeat_Payload > >` [heartbeat_payload_reader](#)

Additional Inherited Members

6.7.1 Detailed Description

Handles Async heartbeat Response.

Definition at line 79 of file `master.h`.

6.7.2 Field Documentation

6.7.2.1 heartbeat_payload_reader

```
std::unique_ptr<grpc::ClientAsyncResponseReader<masterworker::Heartbeat_Payload> > HeartbeatCall::heartbeat_payload_reader
```

Definition at line 83 of file master.h.

6.7.2.2 result

```
masterworker::Heartbeat_Payload HeartbeatCall::result
```

Definition at line 82 of file master.h.

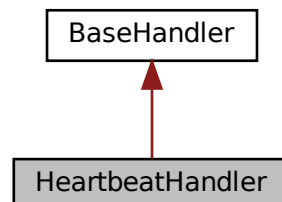
The documentation for this class was generated from the following file:

- [src/master.h](#)

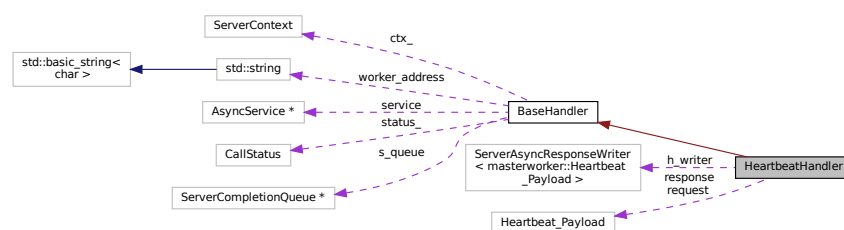
6.8 HeartbeatHandler Class Reference

```
#include <worker.h>
```

Inheritance diagram for HeartbeatHandler:



Collaboration diagram for HeartbeatHandler:



Public Member Functions

- [HeartbeatHandler](#) (masterworker::Map_Reduce::AsyncService *[service](#), grpc::ServerCompletionQueue *pQueue, std::string basicString)
- void [Proceed](#) ()

Private Member Functions

- masterworker::Heartbeat_Payload [handle_heartbeat_job](#) (masterworker::Heartbeat_Payload [request](#))

Private Attributes

- masterworker::Heartbeat_Payload [request](#)
- masterworker::Heartbeat_Payload [response](#)
- grpc::ServerAsyncResponseWriter< masterworker::Heartbeat_Payload > [h_writer](#)

Additional Inherited Members

6.8.1 Detailed Description

Heartbeat class

Definition at line 168 of file worker.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 HeartbeatHandler()

```
HeartbeatHandler::HeartbeatHandler (
    masterworker::Map_Reduce::AsyncService * service,
    grpc::ServerCompletionQueue * pQueue,
    std::string basicString ) [inline]
```

Constructor for Heartbeat class

Parameters

<i>service</i>	
<i>pQueue</i>	
<i>basicString</i>	

Definition at line 177 of file worker.h.

```
181         : BaseHandler(service, pQueue, basicString)
182         , h\_writer(&ctx_)
183     {
```

```

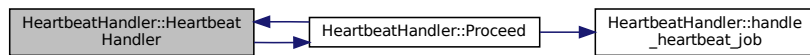
184         HeartbeatHandler::Proceed();
185     }

```

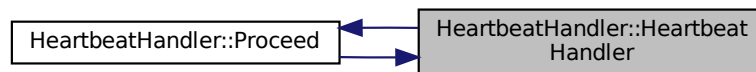
References Proceed().

Referenced by Proceed().

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.3 Member Function Documentation

6.8.3.1 handle_heartbeat_job()

```

masterworker::Heartbeat_Payload HeartbeatHandler::handle_heartbeat_job (
    masterworker::Heartbeat_Payload request ) [inline], [private]

```

Handles Heartbeat request and return heartbeat payload

Parameters

<i>request</i>	Heartbeat payload check .proto
----------------	--------------------------------

Returns

Heartbeat payload

Definition at line 432 of file worker.h.

```

433 {
434     masterworker::Heartbeat_Payload payload;
435     payload.set_id(request.id());
436     if (true)
437         payload.set_status(masterworker::Heartbeat_Payload_type_ALIVE);

```

```

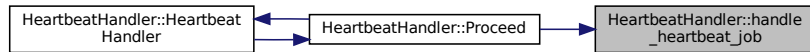
438     return payload;
439 }

```

References request.

Referenced by Proceed().

Here is the caller graph for this function:



6.8.3.2 Proceed()

```
void HeartbeatHandler::Proceed ( ) [inline], [virtual]
```

Reimplemented from [BaseHandler](#).

Definition at line 187 of file worker.h.

```

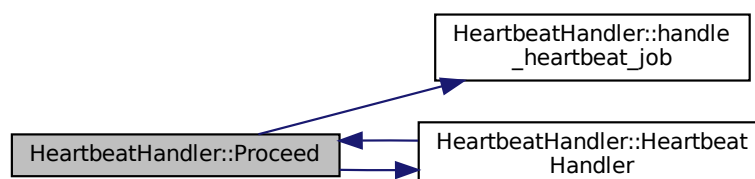
188 {
189     if (status_ == CREATE)
190     {
191         status_ = PROCESS;
192         service->Requestheartbeat(&ctx_, &request, &h_writer, s_queue, s_queue, this);
193     }
194     else if (status_ == PROCESS)
195     {
196         new HeartbeatHandler(service, s_queue, worker_address);
197         response = handle_heartbeat_job(request);
198         status_ = FINISH;
199         h_writer.Finish(response, grpc::Status::OK, this);
200     }
201     else
202     {
203         GPR_ASSERT(status_ == FINISH);
204         delete this;
205     }
206 }

```

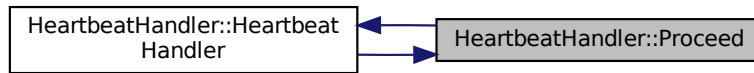
References BaseHandler::CREATE, BaseHandler::ctx_, BaseHandler::FINISH, h_writer, handle_heartbeat_job(), HeartbeatHandler(), BaseHandler::PROCESS, request, response, BaseHandler::s_queue, BaseHandler::service, BaseHandler::status_, and BaseHandler::worker_address.

Referenced by HeartbeatHandler().

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.4 Field Documentation

6.8.4.1 h_writer

```
grpc::ServerAsyncResponseWriter<masterworker::Heartbeat_Payload> HeartbeatHandler::h_writer  
[private]
```

Definition at line 210 of file worker.h.

Referenced by Proceed().

6.8.4.2 request

```
masterworker::Heartbeat_Payload HeartbeatHandler::request [private]
```

Definition at line 209 of file worker.h.

Referenced by handle_heartbeat_job(), and Proceed().

6.8.4.3 response

```
masterworker::Heartbeat_Payload HeartbeatHandler::response [private]
```

Definition at line 209 of file worker.h.

Referenced by Proceed().

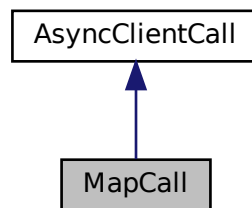
The documentation for this class was generated from the following file:

- [src/worker.h](#)

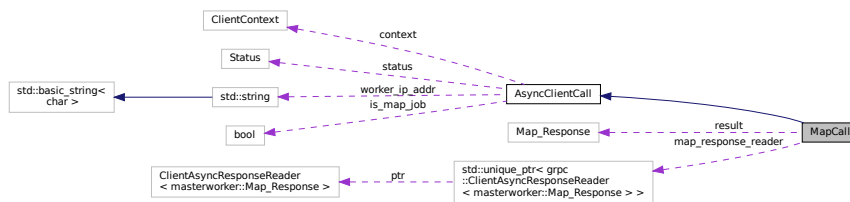
6.9 MapCall Class Reference

```
#include <master.h>
```

Inheritance diagram for MapCall:



Collaboration diagram for MapCall:



Data Fields

- `masterworker::Map_Response` [result](#)
- `std::unique_ptr< grpc::ClientAsyncResponseReader< masterworker::Map_Response > >` [map_response_reader](#)

Additional Inherited Members

6.9.1 Detailed Description

Handles Async Map Response.

Definition at line 61 of file `master.h`.

6.9.2 Field Documentation

6.9.2.1 map_response_reader

```
std::unique_ptr<grpc::ClientAsyncResponseReader<masterworker::Map_Response> > MapCall::map_response_reader
```

Definition at line 65 of file master.h.

6.9.2.2 result

```
masterworker::Map_Response MapCall::result
```

Definition at line 64 of file master.h.

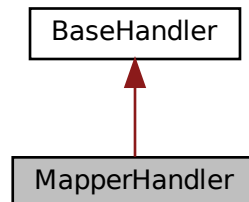
The documentation for this class was generated from the following file:

- [src/master.h](#)

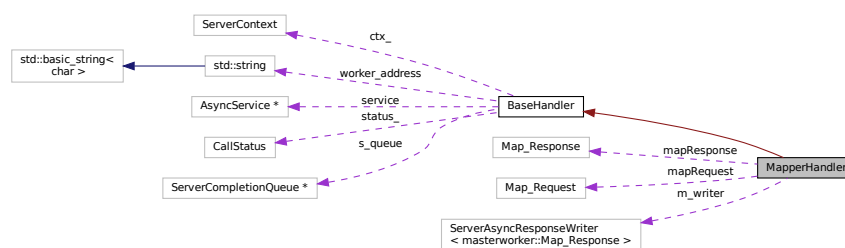
6.10 MapperHandler Class Reference

```
#include <worker.h>
```

Inheritance diagram for MapperHandler:



Collaboration diagram for MapperHandler:



Public Member Functions

- [MapperHandler](#) (masterworker::Map_Reduce::AsyncService *[service](#), grpc::ServerCompletionQueue *p← Queue, std::string [basicString](#))
- void [Proceed](#) ()

Private Member Functions

- masterworker::Map_Response [handle_mapper_job](#) (masterworker::Map_Request request)
- [BaseMapperInternal](#) * [get_basemapper_internal](#) (BaseMapper *mapper)
- [FileShard](#) [convert_grpc_spec](#) (masterworker::partition partition)

Private Attributes

- masterworker::Map_Request [mapRequest](#)
- masterworker::Map_Response [mapResponse](#)
- grpc::ServerAsyncResponseWriter< masterworker::Map_Response > [m_writer](#)

Additional Inherited Members

6.10.1 Detailed Description

Mapper Class

Definition at line 63 of file worker.h.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 MapperHandler()

```
MapperHandler::MapperHandler (
    masterworker::Map_Reduce::AsyncService * service,
    grpc::ServerCompletionQueue * pQueue,
    std::string basicString ) [inline]
```

Constructor for Mapper Class which inits class

Parameters

<i>service</i>	
<i>pQueue</i>	
<i>basicString</i>	

Definition at line 72 of file worker.h.

```

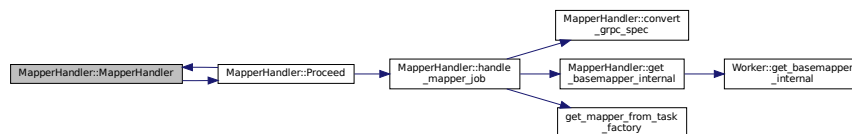
76         : BaseHandler(service, pQueue, basicString)
77         , m_writer(&ctx_)
78     {
79         Proceed();
80     }

```

References Proceed().

Referenced by Proceed().

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3 Member Function Documentation

6.10.3.1 convert_grpc_spec()

```

FileShard MapperHandler::convert_grpc_spec (
    masterworker::partition partition ) [private]

```

Conver grpc payload to [FileShard](#).

Parameters

<i>partition</i>	
------------------	--

Returns

[FileShard](#) struct equivalent of grpc partition payload.

Definition at line 325 of file worker.h.

```

326 {

```

```

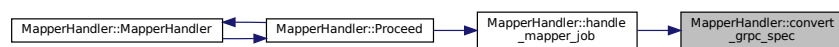
327     FileShard shard{};
328     shard.shard_id = partition.shard_id();
329     for (auto f : partition.file_list())
330     {
331         splitFile temp{};
332         temp.filename = f.filename();
333         temp.offsets = {f.start_offset(), f.end_offset()};
334         shard.split_file_list.push_back(temp);
335     }
336     return shard;
337 }

```

References splitFile::filename, and FileShard::shard_id.

Referenced by handle_mapper_job().

Here is the caller graph for this function:



6.10.3.2 get_basemapper_internal()

```

BaseMapperInternal * MapperHandler::get_basemapper_internal (
    BaseMapper * mapper ) [inline], [private]

```

Parameters

<i>mapper</i>	
---------------	--

Returns

BaseMapperinternal Class

Definition at line 454 of file worker.h.

```

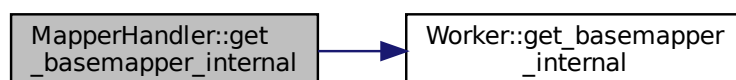
455 {
456     return Worker::get_basemapper_internal(mapper);
457 }

```

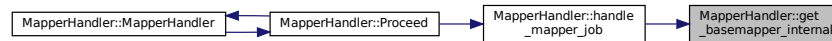
References Worker::get_basemapper_internal().

Referenced by handle_mapper_job().

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.3 handle_mapper_job()

```

masterworker::Map_Response MapperHandler::handle_mapper_job (
    masterworker::Map_Request request ) [inline], [private]
  
```

Given GRPC Map request , Select intermediate file. usually TEMP_DIR/<partition_count>_<worker_port>.txt
Reads shard files and apply user defined map function on it.

Parameters

<i>request</i>	grpc Map Request , check .proto
----------------	---------------------------------

Returns

grpc Map Response payload , check .proto

Definition at line 344 of file worker.h.

```

345 {
346     masterworker::Map_Response payload;
347     auto user_mapper_func = get_mapper_from_task_factory(request.uuid());
348     auto base_mapper = get_basemapper_internal(user_mapper_func.get());
349     auto partition_count = request.partition_count();
350     base_mapper->intermediate_file_list.reserve(partition_count);
351     for (int i = 0; i < partition_count; i++)
352     {
353         base_mapper->intermediate_file_list.push_back(std::string(
354             std::string(TEMP_DIR) + "/" + std::to_string(i) + "_" + MapperHandler::worker_address +
355             ".txt"));
356         FileShard local_shard;
357         for (int shard_count = 0; shard_count < request.shard_size(); shard_count++)
358         {
359             local_shard = MapperHandler::convert_grpc_spec(request.shard(shard_count));
360             for (const auto& i : local_shard.split_file_list)
361             {
362                 std::string mapper_line;
363                 std::ifstream f(i.filename, std::ios::binary);
364                 if (!f.good())
365                 {
366                     std::cerr << i.filename << " not open..." << std::endl;
367                 }
368                 f.seekg(i.offsets.first);
369                 std::string dummy(i.offsets.second - i.offsets.first, ' ');
370                 f.read(&dummy[0], i.offsets.second - i.offsets.first);
371                 std::stringstream stream(dummy);
372                 while (std::getline(stream, mapper_line))
373                 {
374                     user_mapper_func->map(mapper_line);
375                 }
376             }
377         }
378     }
379     base_mapper->final_flush();
380     for (const auto& i : base_mapper->intermediate_file_list)
381     {
382
  
```

```

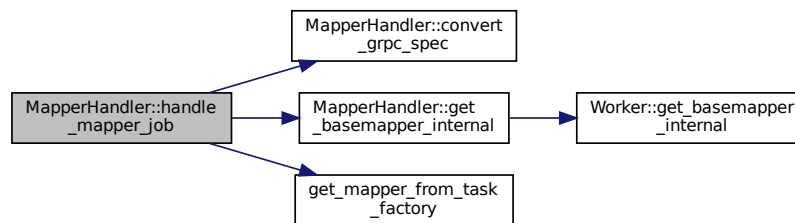
383     payload.add_file_list(i);
384 }
385
386     return payload;
387 }

```

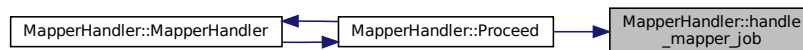
References `convert_grpc_spec()`, `get_basemapper_internal()`, `get_mapper_from_task_factory()`, `FileShard::split_file_list`, `TEMP_DIR`, and `BaseHandler::worker_address`.

Referenced by `Proceed()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.4 Proceed()

```
void MapperHandler::Proceed ( ) [inline], [virtual]
```

Reimplemented from [BaseHandler](#).

Definition at line 82 of file `worker.h`.

```

83     {
84         if (status_ == CREATE)
85         {
86             status_ = PROCESS;
87             service->Requestmap(&ctx_, &mapRequest, &m_writer, s_queue, s_queue, this);
88         }
89         else if (status_ == PROCESS)
90         {
91             new MapperHandler(service, s_queue, worker_address);
92             mapResponse = handle_mapper_job(mapRequest);
93             status_ = FINISH;
94             m_writer.Finish(mapResponse, grpc::Status::OK, this);
95         }
96         else
97         {
98             GPR_ASSERT(status_ == FINISH);
99             delete this;
100         }

```

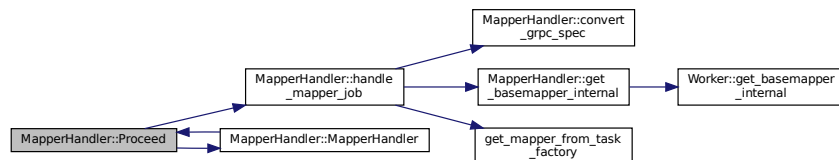


```
101     }
```

References BaseHandler::CREATE, BaseHandler::ctx_, BaseHandler::FINISH, handle_mapper_job(), m_writer, MapperHandler(), mapRequest, mapResponse, BaseHandler::PROCESS, BaseHandler::s_queue, BaseHandler::service, BaseHandler::status_, and BaseHandler::worker_address.

Referenced by MapperHandler().

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.4 Field Documentation

6.10.4.1 m_writer

```
grpc::ServerAsyncResponseWriter<masterworker::Map_Response> MapperHandler::m_writer [private]
```

Definition at line 106 of file worker.h.

Referenced by Proceed().

6.10.4.2 mapRequest

```
masterworker::Map_Request MapperHandler::mapRequest [private]
```

Definition at line 104 of file worker.h.

Referenced by Proceed().

6.10.4.3 mapResponse

```
masterworker::Map_Response MapperHandler::mapResponse [private]
```

Definition at line 105 of file worker.h.

Referenced by Proceed().

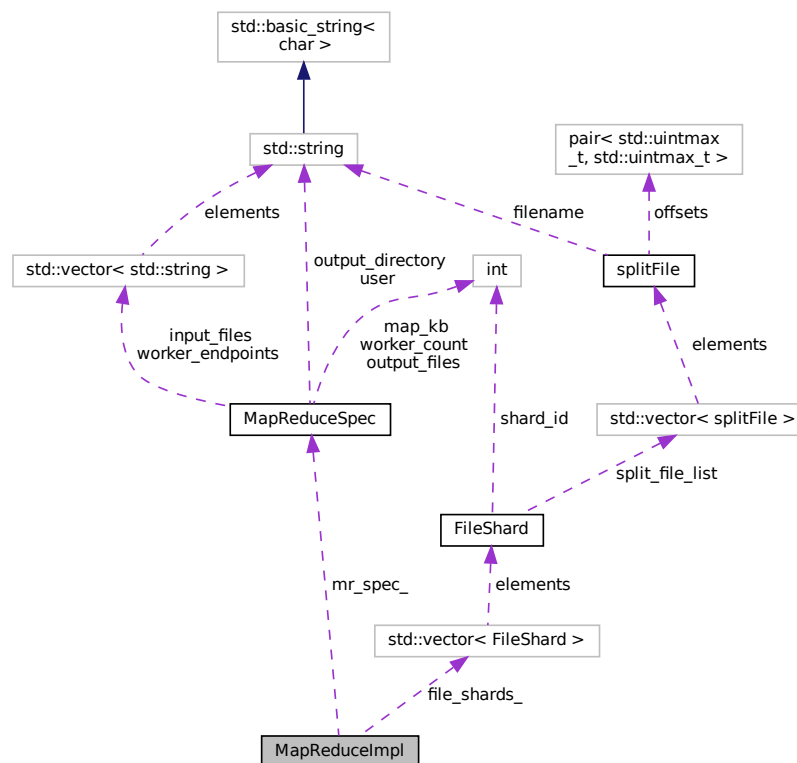
The documentation for this class was generated from the following file:

- [src/worker.h](#)

6.11 MapReduceImpl Class Reference

```
#include <mapreduce_impl.h>
```

Collaboration diagram for MapReduceImpl:



Public Member Functions

- `bool` [run](#) (const `std::string` &config_filename)

Private Member Functions

- bool [read_and_validate_spec](#) (const std::string &config_filename)
- bool [create_shards](#) ()
- bool [run_master](#) ()

Private Attributes

- [MapReduceSpec](#) [mr_spec_](#)
- std::vector< [FileShard](#) > [file_shards_](#)

6.11.1 Detailed Description

Definition at line 5 of file `mapreduce_impl.h`.

6.11.2 Member Function Documentation

6.11.2.1 `create_shards()`

```
bool MapReduceImpl::create_shards ( ) [private]
```

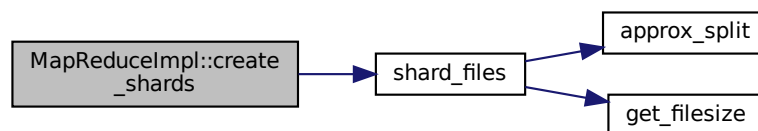
Definition at line 39 of file `mapreduce_impl.cc`.

```
40 {
41     return shard_files(mr_spec_, file_shards_);
42 }
```

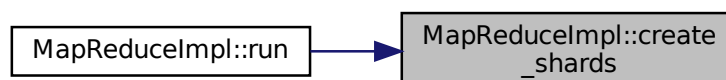
References `file_shards_`, `mr_spec_`, and `shard_files()`.

Referenced by `run()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.2.2 read_and_validate_spec()

```
bool MapReduceImpl::read_and_validate_spec (
    const std::string & config_filename ) [private]
```

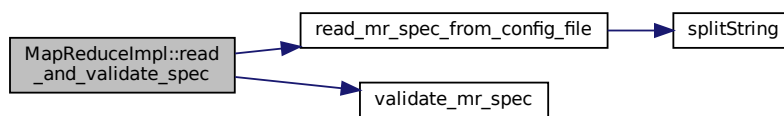
Definition at line 33 of file mapreduce_impl.cc.

```
34 {
35     return read_mr_spec_from_config_file(config_filename, mr_spec_) && validate_mr_spec(mr_spec_);
36 }
```

References `mr_spec_`, `read_mr_spec_from_config_file()`, and `validate_mr_spec()`.

Referenced by `run()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.2.3 run()

```
bool MapReduceImpl::run (
    const std::string & config_filename )
```

Definition at line 8 of file mapreduce_impl.cc.

```
9 {
10
11     if (!read_and_validate_spec(config_filename))
12     {
13         std::cerr << "Spec not configured properly." << std::endl;
14         return false;
15     }
16
17     if (!create_shards())
18     {
19         std::cerr << "Failed to create shards." << std::endl;
20         return false;
21     }
22 }
```

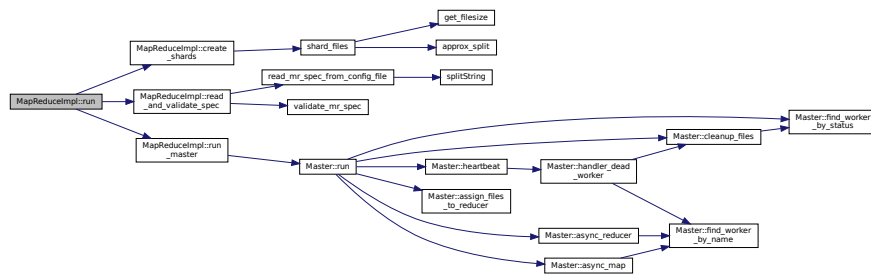
```

22
23     if (!run_master())
24     {
25         std::cerr << "MapReduce failure. Something didn't go well!" << std::endl;
26         return false;
27     }
28
29     return true;
30 }

```

References `create_shards()`, `read_and_validate_spec()`, and `run_master()`.

Here is the call graph for this function:



6.11.2.4 run_master()

```
bool MapReduceImpl::run_master ( ) [private]
```

Definition at line 45 of file `mapreduce_impl.cc`.

```

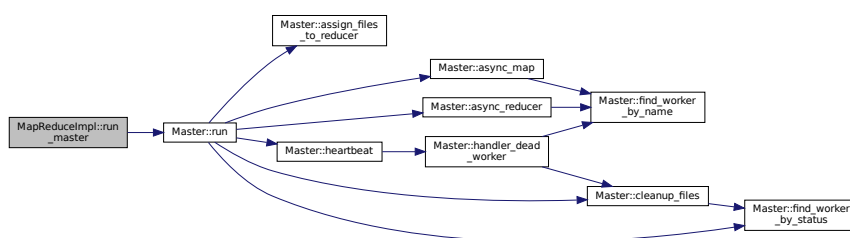
46 {
47     Master master(mr_spec_, file_shards_);
48     return master.run();
49 }

```

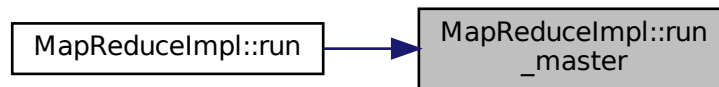
References `file_shards_`, `mr_spec_`, and `Master::run()`.

Referenced by `run()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3 Field Documentation

6.11.3.1 `file_shards_`

```
std::vector<FileShard> MapReduceImpl::file_shards_ [private]
```

Definition at line 19 of file `mapreduce_impl.h`.

Referenced by `create_shards()`, and `run_master()`.

6.11.3.2 `mr_spec_`

```
MapReduceSpec MapReduceImpl::mr_spec_ [private]
```

Definition at line 18 of file `mapreduce_impl.h`.

Referenced by `create_shards()`, `read_and_validate_spec()`, and `run_master()`.

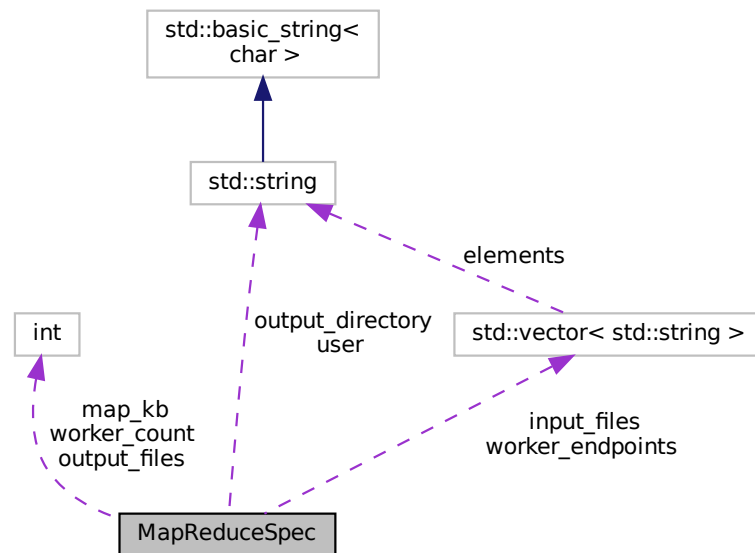
The documentation for this class was generated from the following files:

- [src/mapreduce_impl.h](#)
- [src/mapreduce_impl.cc](#)

6.12 MapReduceSpec Struct Reference

```
#include <mapreduce_spec.h>
```

Collaboration diagram for MapReduceSpec:



Data Fields

- unsigned int `worker_count` = 0
- unsigned int `output_files` = 0
- unsigned int `map_kb` = 0
- `std::string` `user`
- `std::string` `output_directory`
- `std::vector< std::string >` `worker_endpoints`
- `std::vector< std::string >` `input_files`

6.12.1 Detailed Description

Definition at line 23 of file `mapreduce_spec.h`.

6.12.2 Field Documentation

6.12.2.1 input_files

```
std::vector<std::string> MapReduceSpec::input_files
```

Definition at line 31 of file `mapreduce_spec.h`.

Referenced by `read_mr_spec_from_config_file()`, `shard_files()`, and `validate_mr_spec()`.

6.12.2.2 map_kb

```
unsigned int MapReduceSpec::map_kb = 0
```

Definition at line 27 of file `mapreduce_spec.h`.

Referenced by `read_mr_spec_from_config_file()`, and `shard_files()`.

6.12.2.3 output_directory

```
std::string MapReduceSpec::output_directory
```

Definition at line 29 of file `mapreduce_spec.h`.

Referenced by `read_mr_spec_from_config_file()`, `Master::run()`, and `validate_mr_spec()`.

6.12.2.4 output_files

```
unsigned int MapReduceSpec::output_files = 0
```

Definition at line 26 of file `mapreduce_spec.h`.

Referenced by `read_mr_spec_from_config_file()`, `Master::run()`, and `WorkerClient::schedule_mapper_jobs()`.

6.12.2.5 user

```
std::string MapReduceSpec::user
```

Definition at line 28 of file `mapreduce_spec.h`.

Referenced by `read_mr_spec_from_config_file()`, `WorkerClient::schedule_mapper_jobs()`, and `WorkerClient::schedule_reduce_job()`.


```
unsigned int MapReduceSpec::worker_count = 0
```

Referenced by `read_mr_spec_from_config_file()`, and `validate_mr_spec()`.

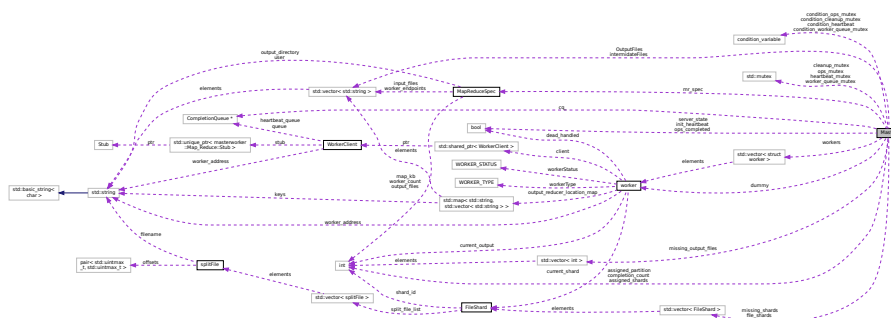
```
std::vector<std::string> MapReduceSpec::worker_endpoints
```

Referenced by `read_mr_spec_from_config_file()`, and `validate_mr_spec()`.

- `src/mapreduce_spec.h`

```
#include <master.h>
```

Collaboration diagram for Master:



- `Master` (const `MapReduceSpec` &, const `std::vector< FileShard >` &)
- `bool run ()`
- `~Master ()`

Private Member Functions

- `worker * find_worker_by_name (std::string t)`
- `std::vector< int > find_worker_by_status (WORKER_STATUS t)`
- `void heartbeat ()`
- `void handler_dead_worker (std::string worker)`
- `void cleanup_files ()`
- `void async_map ()`
- `void async_reducer ()`
- `std::vector< std::string > assign_files_to_reducer (int output_id)`

Private Attributes

- `grpc::CompletionQueue * cq_`
- `bool server_state = true`
- `MapReduceSpec mr_spec`
- `worker dummy {}`
- `std::vector< struct worker > workers {}`
- `std::mutex worker_queue_mutex`
- `std::condition_variable condition_worker_queue_mutex`
- `bool init_heartbeat = true`
- `std::mutex heartbeat_mutex`
- `std::condition_variable condition_heartbeat`
- `std::mutex cleanup_mutex`
- `std::condition_variable condition_cleanup_mutex`
- `int completion_count`
- `bool ops_completed = false`
- `std::mutex ops_mutex`
- `std::condition_variable condition_ops_mutex`
- `int assigned_shards`
- `std::vector< FileShard > file_shards`
- `std::vector< FileShard > missing_shards`
- `std::vector< std::string > intermidateFiles`
- `int assigned_partition`
- `std::vector< std::string > OutputFiles`
- `std::vector< int > missing_output_files`

6.13.1 Detailed Description

Definition at line 253 of file master.h.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 Master()

```
Master::Master (
    const MapReduceSpec & mr_spec,
    const std::vector< FileShard > & file_shards )
```

Constructor for `Master` , Inits worker clients given spec file. This is all the information your master will get from the framework. You can populate your other class data members here if you want

Parameters

<i>mr_spec</i>	
<i>file_shards</i>	

Definition at line 363 of file master.h.

```

364     : mr_spec(mr_spec)
365     , file_shards(file_shards)
366 {
367     cq_ = new grpc::CompletionQueue();
368     for (const auto& i : Master::mr_spec.worker_endpoints)
369     {
370         dummy.worker_address = i;
371         dummy.workerStatus = FREE;
372         dummy.workerType = MAPPER;
373         dummy.client = std::make_shared<WorkerClient>(i, Master::cq_);
374         Master::workers.push_back(dummy);
375     }
376 }
```

References worker::client, cq_, dummy, FREE, MAPPER, mr_spec, worker::worker_address, workers, worker↵
::workerStatus, and worker::workerType.

6.13.2.2 ~Master()

```
Master::~~Master ( ) [inline]
```

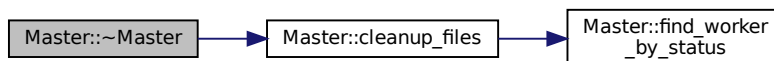
Definition at line 262 of file master.h.

```

263     {
264         Master::server_state = !ALIVE;
265         Master::cq_->Shutdown();
266         cleanup_files();
267     }
```

References ALIVE, cleanup_files(), cq_, and server_state.

Here is the call graph for this function:



6.13.3 Member Function Documentation

6.13.3.1 assign_files_to_reducer()

```
std::vector< std::string > Master::assign_files_to_reducer (
    int output_id ) [private]
```

Assigns list of intermediate files to output files for reducing

Parameters

<i>output</i> ↔	
<i>_id</i>	

Returns

list of intermediate files for given output id /file

Definition at line 815 of file master.h.

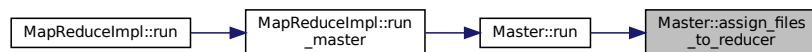
```

816 {
817     std::set<std::string> file_list;
818     for (int i = 0; i < Master::intermidateFiles.size(); i++)
819     {
820         auto f = Master::intermidateFiles[i];
821         if (i % Master::mr_spec.output_files == output_id)
822         {
823             file_list.insert(f);
824         }
825     }
826     std::vector<std::string> convert;
827     convert.assign(file_list.begin(), file_list.end());
828     return convert;
829 }
```

References intermidateFiles, and mr_spec.

Referenced by run().

Here is the caller graph for this function:



6.13.3.2 async_map()

```
void Master::async_map ( ) [private]
```

Handles Async Responses for Mapper Requests and frees worker for other work and puts data back in in the list

Definition at line 706 of file master.h.

```

707 {
708     void* tag;
709     bool ok = false;
710     while (Master::cq_>Next(&tag, &ok))
711     {
712         auto call = static_cast<AsyncClientCall*>(tag);
713         if (call->status.ok())
714         {
715             if (Master::find_worker_by_name(call->worker_ip_addr)->workerStatus != DEAD)
716             {
717                 {
718                     std::lock_guard<std::mutex> worker_queue(this->worker_queue_mutex);
719                     for (auto& worker : Master::workers)
720                     {
721                         if (worker.worker_address == call->worker_ip_addr)
722                         {
723                             std::cout << call->worker_ip_addr + " back to free." << std::endl;
```

```

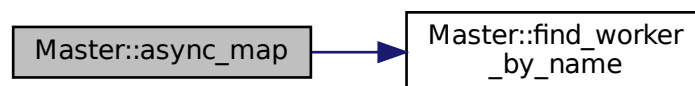
724
725         worker.workerStatus = FREE;
726         Master::completion_count--;
727         std::cout << call->worker_ip_addr + " response recieved. Completion Count : "
+
728         std::to_string(Master::completion_count) +
729         " Assigned Work: " +
std::to_string(Master::assigned_shards)
730         << std::endl;
731         break;
732     }
733 }
734 condition_worker_queue_mutex.notify_one();
735 }
736 if (call->is_map_job)
737 {
738     auto mcall = dynamic_cast<MapCall*>(call);
739     for (const auto& m : mcall->result.file_list())
740     {
741         Master::intermidateFiles.push_back(m);
742     }
743 }
744 {
745     std::unique_lock<std::mutex> work_done(ops_mutex);
746     if (Master::completion_count == 0)
747     {
748         ops_completed = true;
749         condition_ops_mutex.notify_one();
750         break;
751     }
752 }
753 }
754 condition_cleanup_mutex.notify_one();
755 }
756 delete call;
757 }
758 }

```

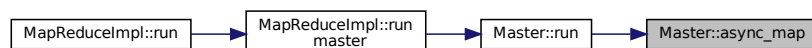
References assigned_shards, completion_count, condition_cleanup_mutex, condition_ops_mutex, condition_worker_queue_mutex, cq_, DEAD, find_worker_by_name(), FREE, intermidateFiles, ops_completed, ops_mutex, worker::worker_address, worker_queue_mutex, workers, and worker::workerStatus.

Referenced by run().

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.3 async_reducer()

```
void Master::async_reducer ( ) [private]
```

Similar Async_map , handles Reducer Responses.

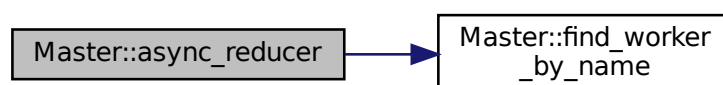
Definition at line 762 of file master.h.

```
763 {
764     void* tag;
765     bool ok = false;
766     while (Master::cq_>Next(&tag, &ok))
767     {
768         auto call = static_cast<AsyncClientCall*>(tag);
769         if (call->status.ok())
770         {
771             if (Master::find_worker_by_name(call->worker_ip_addr)->workerStatus != DEAD)
772             {
773                 {
774                     std::lock_guard<std::mutex> worker_queue(this->worker_queue_mutex);
775                     for (auto& worker : Master::workers)
776                     {
777                         if (worker.worker_address == call->worker_ip_addr)
778                         {
779                             worker.workerStatus = FREE;
780                             Master::completion_count--;
781                             std::cout << call->worker_ip_addr + " response received. Completion Count : "
782                                     + std::to_string(Master::completion_count) +
783                                     " Assigned Work: " +
784                                     std::to_string(Master::assigned_partition)
785                                     << std::endl;
786                             break;
787                         }
788                     }
789                     condition_worker_queue_mutex.notify_one();
790                 }
791                 if (!call->is_map_job)
792                 {
793                     auto mcall = dynamic_cast<ReduceCall*>(call);
794                     Master::OutputFiles.push_back(mcall->result.file_name());
795                 }
796                 {
797                     std::unique_lock<std::mutex> work_done(ops_mutex);
798                     if (Master::completion_count == 0)
799                     {
800                         ops_completed = true;
801                         condition_ops_mutex.notify_one();
802                         break;
803                     }
804                 }
805                 condition_cleanup_mutex.notify_one();
806             }
807             delete call;
808         }
809     }
```

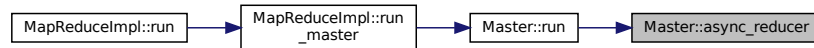
References assigned_partition, completion_count, condition_cleanup_mutex, condition_ops_mutex, condition_↔_worker_queue_mutex, cq_, DEAD, find_worker_by_name(), FREE, ops_completed, ops_mutex, OutputFiles, worker::worker_address, worker_queue_mutex, workers, and worker::workerStatus.

Referenced by run().

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.4 cleanup_files()

```
void Master::cleanup_files ( ) [private]
```

Cleans up file if Server is alive and if there is any Dead worker in the list. and also handles intermediate files cleanup during master's exit.

Definition at line 653 of file master.h.

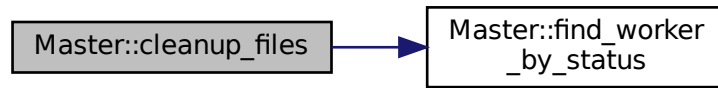
```

654 {
655     if (!Master::find_worker_by_status(DEAD).empty() && Master::server_state == ALIVE)
656     {
657         for (auto i : Master::find_worker_by_status(DEAD))
658         {
659             if (Master::workers[i].workerType == MAPPER && !Master::workers[i].dead_handled)
660             {
661                 // std::string(TEMP_DIR) + "/" + std::to_string(i) + "_" + MapperHandler::worker_address
662                 + ".txt"))
663                 auto worker_port =
664                     Master::workers[i].worker_address.substr(Master::workers[i].worker_address.find_first_of(':'));
665 #if __cplusplus >= 201703L
666                 for (auto f : fs::directory_iterator(TEMP_DIR))
667                 {
668                     if (f.path().string().find(worker_port) != std::string::npos)
669                         fs::remove(f);
670                 }
671 #else
672                 auto dir_string = std::string("rm -rf ") + TEMP_DIR + "/*_" + worker_port + ".txt";
673                 system(dir_string.c_str());
674 #endif
675             }
676         }
677     }
678     else
679     {
680         for (const auto& worker_location : Master::workers[i].output_reducer_location_map)
681         {
682             if (!Master::workers[i].dead_handled)
683             #if __cplusplus >= 201703L
684                 fs::remove(worker_location.first);
685             #else
686                 remove(worker_location.first.c_str());
687             #endif
688         }
689         Master::workers[i].dead_handled = true;
690     }
691 }
692 }
693 if (!Master::server_state)
694 {
695 #if __cplusplus >= 201703L
696     fs::remove_all(TEMP_DIR);
697 #else
698     auto dir_string = std::string("rm -rf ") + TEMP_DIR;
699     system(dir_string.c_str());
700 #endif
701 }
702 }
```

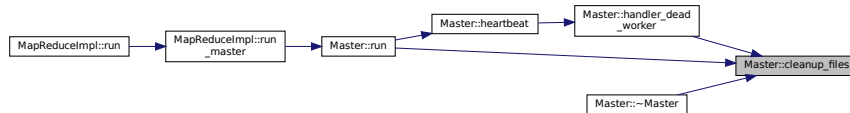
References ALIVE, DEAD, find_worker_by_status(), MAPPER, server_state, TEMP_DIR, and workers.

Referenced by `handler_dead_worker()`, `run()`, and `~Master()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.5 find_worker_by_name()

```

worker * Master::find_worker_by_name (
    std::string t ) [private]
  
```

Return [Worker](#) id with requested Name,

Parameters

<i>t</i>	
----------	--

Returns

pointer for [Worker](#) with requested name or Null pointer if not found

Definition at line 343 of file `master.h`.

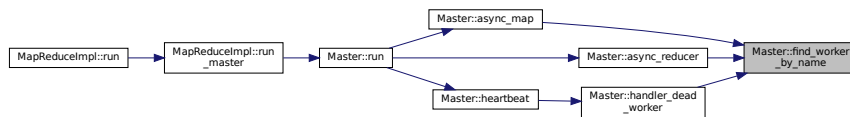
```

344 {
345     for (auto& w : Master::workers)
346     {
347         if (w.worker_address == t)
348             return &w;
349     }
350     std::cerr << "worker " << t << " not found" << std::endl;
351     return nullptr;
352 }
  
```

References `workers`.

Referenced by `async_map()`, `async_reducer()`, and `handler_dead_worker()`.

Here is the caller graph for this function:



6.13.3.6 find_worker_by_status()

```
std::vector< int > Master::find_worker_by_status (
    WORKER_STATUS t ) [private]
```

Return list of [Worker](#) id with requested status, FREE OR BUSY OR DEAD

Parameters

<i>t</i>	
----------	--

Returns

list of [Worker](#) id with requested status

Definition at line 325 of file master.h.

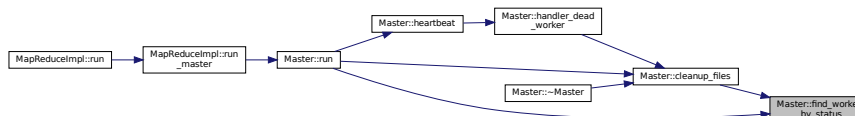
```

326 {
327     std::vector<int> temp;
328     for (int i = 0; i < Master::workers.size(); i++)
329     {
330         if (Master::workers[i].workerStatus == t)
331         {
332             temp.push_back(i);
333         }
334     }
335     return temp;
336 }
```

References `workers`.

Referenced by `cleanup_files()`, and `run()`.

Here is the caller graph for this function:



6.13.3.7 handler_dead_worker()

```
void Master::handler_dead_worker (
    std::string worker ) [private]
```

Given worker ,Cleans up half ass work for the worker and moves the file shards or ouput files back to pool for assigning to other workers

Parameters

<i>worker</i>	address/id
---------------	------------

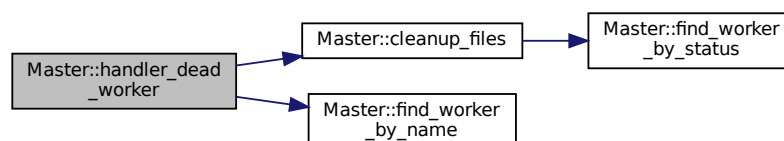
Definition at line 611 of file master.h.

```
612 {
613     std::cerr << "HANDLING DEAD WORKER....." + worker << std::endl;
614     auto w = Master::find_worker_by_name(worker);
615     // Handle Mapper
616     if (w->workerType == MAPPER and !ops_completed)
617     {
618         std::lock_guard<std::mutex> lockGuard(Master::cleanup_mutex);
619         auto c = w->client.get();
620         // uninitialized....
621         if (!c)
622         {
623             condition_ops_mutex.notify_all();
624             return;
625         }
626         Master::missing_shards.push_back(w->current_shard);
627         Master::assigned_shards++;
628         w->workerStatus = DEAD;
629         Master::cleanup_files();
630         condition_cleanup_mutex.notify_one();
631     }
632     else
633     {
634         std::lock_guard<std::mutex> lockGuard(Master::cleanup_mutex);
635         auto c = w->client.get();
636         if (!c)
637         {
638             condition_ops_mutex.notify_all();
639             return;
640         }
641         Master::missing_output_files.push_back(w->current_output);
642         Master::assigned_partition++;
643         w->workerStatus = DEAD;
644         Master::cleanup_files();
645         condition_cleanup_mutex.notify_one();
646     }
647     condition_ops_mutex.notify_all();
648 }
```

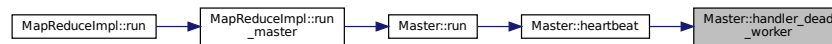
References assigned_partition, assigned_shards, cleanup_files(), cleanup_mutex, condition_cleanup_mutex, condition_ops_mutex, DEAD, find_worker_by_name(), MAPPER, missing_output_files, missing_shards, and ops_completed.

Referenced by heartbeat().

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.8 heartbeat()

```
void Master::heartbeat ( ) [private]
```

Handles heartbeat checks

1. While server is alive, sends heartbeat messages Asyncly and waits for response.
2. If Response timesout or comes as dead calls cleanup. function and re assigns the work.

Definition at line 550 of file master.h.

```

551 {
552     while (Master::server_state)
553     {
554         std::map<std::string, heartbeat_payload> message_queue;
555         auto current_time = std::chrono::system_clock::now().time_since_epoch().count();
556         for (const auto& w : Master::workers)
557         {
558             auto c = w.client.get();
559             heartbeat_payload temp_payload{};
560             temp_payload.id = w.worker_address;
561             // temp_payload.timestamp = current_time;
562             if (w.workerStatus != DEAD)
563             {
564                 c->send_heartbeat(temp_payload.timestamp);
565                 message_queue[w.worker_address] = temp_payload;
566             }
567         }
568
569         for (auto& w : this->workers)
570         {
571             auto c = w.client.get();
572             if (!c)
573                 continue;
574             if (w.workerStatus != DEAD)
575             {
576                 bool status = c->recv_heartbeat();
577                 if (!status)
578                 {
579                     std::cerr << "Error " << w.worker_address << " : Dead , cleaning up" << std::endl;
580                     w.workerStatus = DEAD;
581                     Master::handler_dead_worker(message_queue[w.worker_address].id);
582                 }
583             }
584         }
585
586         if (init_heartbeat)
587         {
588             {
589                 std::unique_lock<std::mutex> heartbeat_lock(Master::heartbeat_mutex);
590                 init_heartbeat = false;
591                 condition_heartbeat.notify_one();
592             }
593         }
594
595         auto end_time = std::chrono::system_clock::now().time_since_epoch().count();
596
597         if (end_time - current_time < 1000 * 1000)
598         {
599             std::unique_lock<std::mutex> heartbeat_lock(Master::heartbeat_mutex);
600             sleep(1);

```

```

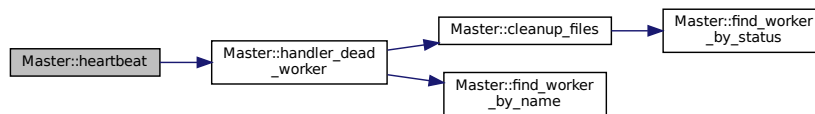
601         condition_heartbeat.wait_for(heartbeat_lock, std::chrono::milliseconds(5 * 1000), [this] {
602             return true; });
603     }
604 }

```

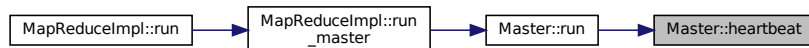
References condition_heartbeat, DEAD, handler_dead_worker(), heartbeat_mutex, heartbeat_payload::id, init_heartbeat, server_state, and workers.

Referenced by run().

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.3.9 run()

```
bool Master::run ( )
```

Brain of code ;) Handles few things

1. Creates Temp intermediate Dir
2. Create Heartbeat thread for monitoring status of workers
3. Assigns Workers Mapper , checks for dead worker and reassigns the work after cleanup.
4. Creates mapping required for intermediate file to Output files.
5. Assigns work to reducers and handle any dead reducers.
6. Handles cleanup for intermediate files

Returns

true false based of worker update.

Definition at line 391 of file master.h.

```

392 {
393     std::thread check_heartbeat_status(&Master::heartbeat, this);
394 #if __cplusplus >= 201703L
395     fs::create_directory(TEMP_DIR);
396     if (fs::is_directory(Master::mr_spec.output_directory))
397     {
398         for (const auto& fi : fs::directory_iterator(Master::mr_spec.output_directory))
399         {
400             fs::remove(fi.path());
401         }
402     }
403 #else
404     auto dir_string = std::string("rm -rf ") + std::string(TEMP_DIR);
405     system(dir_string.c_str());
406     mkdir(TEMP_DIR, 0755);
407 #endif
408
409     std::thread map_job(&Master::async_map, this);
410     {
411         std::unique_lock<std::mutex> lock_heartbeat(heartbeat_mutex);
412         Master::init_heartbeat = true;
413         condition_heartbeat.wait(lock_heartbeat, [this] { return !this->init_heartbeat; });
414     }
415     bool shards_done = false;
416     Master::completion_count = Master::assigned_shards = Master::file_shards.size();
417     while (!shards_done)
418     {
419         for (const auto& s : Master::file_shards)
420         {
421             int i;
422             {
423                 std::unique_lock<std::mutex> shards(Master::cleanup_mutex);
424                 // wait for cleanup mutex and free worker.
425                 condition_cleanup_mutex.wait(shards, [this] { return
!Master::find_worker_by_status(FREE).empty(); });
426                 i = Master::find_worker_by_status(FREE)[0];
427                 if (Master::workers[i].workerStatus == DEAD)
428                 {
429                     continue;
430                 }
431                 Master::workers[i].current_shard = s;
432                 Master::workers[i].workerStatus = BUSY;
433                 Master::assigned_shards--;
434             }
435             auto client = Master::workers[i].client.get();
436             // Adds Mapper Job
437             std::cout << "Assigning Map Work of shard id " + std::to_string(s.shard_id) + " to " +
438                 Master::workers[i].worker_address
439                 << std::endl;
440
441             client->schedule_mapper_jobs(Master::mr_spec, Master::workers[i].current_shard);
442         }
443         {
444             std::unique_lock<std::mutex> work_done(ops_mutex);
445             condition_ops_mutex.wait(work_done);
446             if (assigned_shards <= 0 && ops_completed)
447                 shards_done = true;
448         }
449     }
450     {
451         std::unique_lock<std::mutex> shards(Master::cleanup_mutex);
452         if (Master::assigned_shards > 0 && !Master::missing_shards.empty())
453         {
454             std::cout << "Re assigning work for " << Master::missing_shards[0].shard_id << std::endl;
455             Master::file_shards.clear();
456             Master::file_shards.assign(Master::missing_shards.begin(),
Master::missing_shards.end());
457             Master::missing_shards.clear();
458             condition_cleanup_mutex.notify_one();
459         }
460     }
461 }
462 map_job.join();
463 std::cout << "Map Done." << std::endl;
464
465 for (auto& s : Master::workers)
466 {
467     if (s.workerStatus == DEAD)
468         continue;
469     s.workerStatus = FREE;
470     s.workerType = REDUCER;

```

```

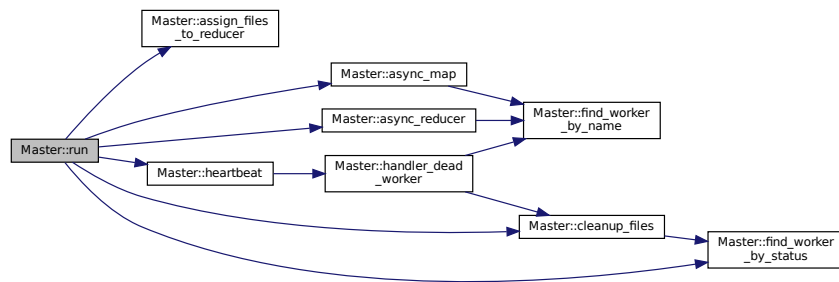
471     }
472     ops_completed = false;
473
474     std::thread reduce_job(&Master::async_reducer, this);
475
476     bool partition_done = false;
477     Master::completion_count = Master::assigned_partition = Master::mr_spec.output_files;
478     std::vector<int> output_vector(Master::assigned_partition);
479     std::iota(output_vector.begin(), output_vector.end(), 0);
480     while (!partition_done && Master::assigned_partition > 0)
481     {
482         for (auto& i : output_vector)
483         {
484             int j;
485             std::string output_file;
486             {
487                 std::unique_lock<std::mutex> partition(Master::cleanup_mutex);
488                 condition_cleanup_mutex.wait(
489                     partition, [this] { return !Master::find_worker_by_status(FREE).empty(); });
490                 j = Master::find_worker_by_status(FREE)[0];
491                 if (Master::workers[j].workerStatus == DEAD)
492                 {
493                     continue;
494                 }
495                 Master::workers[j].workerType = REDUCER;
496                 output_file =
497                     Master::mr_spec.output_directory + "/" +
498                     std::string("output_file_").append(std::to_string(i));
499                 Master::workers[j].output_reducer_location_map[output_file] =
500                     assign_files_to_reducer(i);
501                 Master::workers[j].current_output = i;
502                 Master::workers[j].workerStatus = BUSY;
503                 Master::assigned_partition--;
504             }
505             condition_cleanup_mutex.notify_one();
506             auto client = Master::workers[j].client.get();
507             // Adds Reducer Job
508             std::cout << "Assigning Reduce Work " + output_file + " to " +
509                 Master::workers[j].worker_address
510                 << std::endl;
511             client->schedule_reduce_job(
512                 Master::mr_spec, Master::workers[j].output_reducer_location_map[output_file],
513                 output_file);
514         }
515         {
516             std::unique_lock<std::mutex> work_done(ops_mutex);
517             condition_ops_mutex.wait(work_done);
518             if (Master::assigned_partition <= 0 && ops_completed)
519                 partition_done = true;
520         }
521         {
522             std::unique_lock<std::mutex> partition(Master::cleanup_mutex);
523             if (Master::assigned_partition > 0 && !Master::missing_output_files.empty())
524             {
525                 std::cout << "Re assigning work for " + Master::mr_spec.output_directory + "/" +
526                     std::string("output_file_")
527                     << Master::missing_output_files[0] << std::endl;
528                 output_vector.clear();
529                 output_vector.assign(Master::missing_output_files.begin(),
530                     Master::missing_output_files.end());
531                 Master::missing_output_files.clear();
532                 condition_cleanup_mutex.notify_one();
533             }
534         }
535     }
536     reduce_job.join();
537     {
538         std::unique_lock<std::mutex> heartbeat(Master::heartbeat_mutex);
539         Master::server_state = !ALIVE;
540         condition_heartbeat.notify_all();
541     }
542     check_heartbeat_status.join();
543     cleanup_files();
544     return true;
545 }

```

References ALIVE, assign_files_to_reducer(), assigned_partition, assigned_shards, async_map(), async_reducer(), BUSY, cleanup_files(), cleanup_mutex, completion_count, condition_cleanup_mutex, condition_heartbeat, condition_ops_mutex, DEAD, file_shards, find_worker_by_status(), FREE, heartbeat(), heartbeat_mutex, init_heartbeat, missing_output_files, missing_shards, mr_spec, ops_completed, ops_mutex, MapReduceSpec::output_directory, MapReduceSpec::output_files, REDUCER, server_state, TEMP_DIR, and workers.

Referenced by MapReduceImpl::run_master().

Here is the call graph for this function:



Here is the caller graph for this function:



6.13.4 Field Documentation

6.13.4.1 assigned_partition

```
int Master::assigned_partition [private]
```

Definition at line 312 of file master.h.

Referenced by `async_reducer()`, `handler_dead_worker()`, and `run()`.

6.13.4.2 assigned_shards

```
int Master::assigned_shards [private]
```

Definition at line 305 of file master.h.

Referenced by `async_map()`, `handler_dead_worker()`, and `run()`.

6.13.4.3 cleanup_mutex

```
std::mutex Master::cleanup_mutex [private]
```

Definition at line 294 of file master.h.

Referenced by handler_dead_worker(), and run().

6.13.4.4 completion_count

```
int Master::completion_count [private]
```

Definition at line 299 of file master.h.

Referenced by async_map(), async_reducer(), and run().

6.13.4.5 condition_cleanup_mutex

```
std::condition_variable Master::condition_cleanup_mutex [private]
```

Definition at line 295 of file master.h.

Referenced by async_map(), async_reducer(), handler_dead_worker(), and run().

6.13.4.6 condition_heartbeat

```
std::condition_variable Master::condition_heartbeat [private]
```

Definition at line 289 of file master.h.

Referenced by heartbeat(), and run().

6.13.4.7 condition_ops_mutex

```
std::condition_variable Master::condition_ops_mutex [private]
```

Definition at line 302 of file master.h.

Referenced by async_map(), async_reducer(), handler_dead_worker(), and run().

6.13.4.8 condition_worker_queue_mutex

```
std::condition_variable Master::condition_worker_queue_mutex [private]
```

Definition at line 282 of file master.h.

Referenced by `async_map()`, and `async_reducer()`.

6.13.4.9 cq_

```
grpc::CompletionQueue* Master::cq_ [private]
```

Definition at line 273 of file master.h.

Referenced by `async_map()`, `async_reducer()`, `Master()`, and `~Master()`.

6.13.4.10 dummy

```
worker Master::dummy {} [private]
```

Definition at line 279 of file master.h.

Referenced by `Master()`.

6.13.4.11 file_shards

```
std::vector<FileShard> Master::file_shards [private]
```

Definition at line 306 of file master.h.

Referenced by `run()`.

6.13.4.12 heartbeat_mutex

```
std::mutex Master::heartbeat_mutex [private]
```

Definition at line 288 of file master.h.

Referenced by `heartbeat()`, and `run()`.

6.13.4.13 init_heartbeat

```
bool Master::init_heartbeat = true [private]
```

Definition at line 287 of file master.h.

Referenced by heartbeat(), and run().

6.13.4.14 intermidateFiles

```
std::vector<std::string> Master::intermidateFiles [private]
```

Definition at line 308 of file master.h.

Referenced by assign_files_to_reducer(), and async_map().

6.13.4.15 missing_output_files

```
std::vector<int> Master::missing_output_files [private]
```

Definition at line 314 of file master.h.

Referenced by handler_dead_worker(), and run().

6.13.4.16 missing_shards

```
std::vector<FileShard> Master::missing_shards [private]
```

Definition at line 307 of file master.h.

Referenced by handler_dead_worker(), and run().

6.13.4.17 mr_spec

```
MapReduceSpec Master::mr_spec [private]
```

Definition at line 276 of file master.h.

Referenced by assign_files_to_reducer(), Master(), and run().

6.13.4.18 ops_completed

```
bool Master::ops_completed = false [private]
```

Definition at line 300 of file master.h.

Referenced by `async_map()`, `async_reducer()`, `handler_dead_worker()`, and `run()`.

6.13.4.19 ops_mutex

```
std::mutex Master::ops_mutex [private]
```

Definition at line 301 of file master.h.

Referenced by `async_map()`, `async_reducer()`, and `run()`.

6.13.4.20 OutputFiles

```
std::vector<std::string> Master::OutputFiles [private]
```

Definition at line 313 of file master.h.

Referenced by `async_reducer()`.

6.13.4.21 server_state

```
bool Master::server_state = true [private]
```

Definition at line 274 of file master.h.

Referenced by `cleanup_files()`, `heartbeat()`, `run()`, and `~Master()`.

6.13.4.22 worker_queue_mutex

```
std::mutex Master::worker_queue_mutex [private]
```

Definition at line 281 of file master.h.

Referenced by `async_map()`, and `async_reducer()`.

6.13.4.23 workers

```
std::vector<struct worker> Master::workers {} [private]
```

Definition at line 280 of file master.h.

Referenced by `async_map()`, `async_reducer()`, `cleanup_files()`, `find_worker_by_name()`, `find_worker_by_status()`, `heartbeat()`, `Master()`, and `run()`.

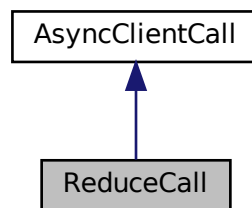
The documentation for this class was generated from the following file:

- `src/master.h`

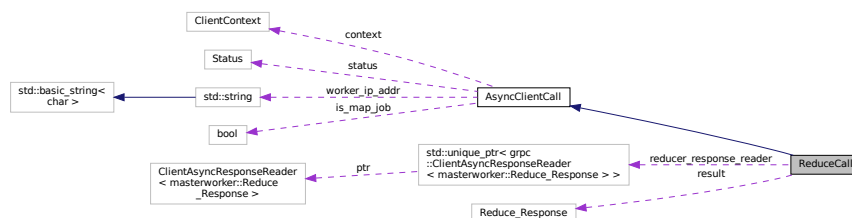
6.14 ReduceCall Class Reference

```
#include <master.h>
```

Inheritance diagram for ReduceCall:



Collaboration diagram for ReduceCall:



Data Fields

- `masterworker::Reduce_Response` [result](#)
- `std::unique_ptr<grpc::ClientAsyncResponseReader<masterworker::Reduce_Response>>` [reducer_response_reader](#)

Additional Inherited Members

6.14.1 Detailed Description

Handles Async Reduce Response.

Definition at line 70 of file master.h.

6.14.2 Field Documentation

6.14.2.1 reducer_response_reader

```
std::unique_ptr<grpc::ClientAsyncResponseReader<masterworker::Reduce_Response> > ReduceCall←  
::reducer_response_reader
```

Definition at line 74 of file master.h.

6.14.2.2 result

```
masterworker::Reduce_Response ReduceCall::result
```

Definition at line 73 of file master.h.

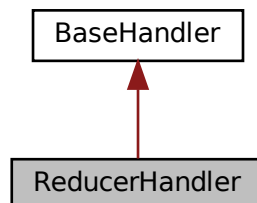
The documentation for this class was generated from the following file:

- [src/master.h](#)

6.15 ReducerHandler Class Reference

```
#include <worker.h>
```

Inheritance diagram for ReducerHandler:



Parameters

<i>service</i>	
<i>pQueue</i>	
<i>basicString</i>	

Definition at line 125 of file worker.h.

```

129         : BaseHandler(service, pQueue, basicString)
130         , r_writer(&ctx_)
131     {
132         ReducerHandler::Proceed();
133     }

```

References `Proceed()`.

Referenced by `Proceed()`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.15.3 Member Function Documentation

6.15.3.1 `get_basereducer_internal()`

```

BaseReducerInternal * ReducerHandler::get_basereducer_internal (
    BaseReducer * reducer ) [inline], [private]

```

Parameters

<i>reducer</i>	
----------------	--

Returns

BaseReducerinternal Class

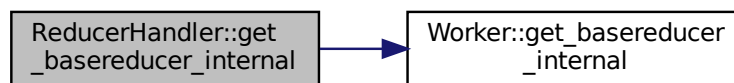
Definition at line 445 of file worker.h.

```
446 {
447     return Worker::get_basereducer_internal(reducer);
448 }
```

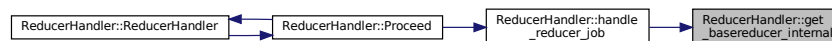
References Worker::get_basereducer_internal().

Referenced by handle_reducer_job().

Here is the call graph for this function:



Here is the caller graph for this function:

**6.15.3.2 handle_reducer_job()**

```
masterworker::Reduce_Response ReducerHandler::handle_reducer_job (
    masterworker::Reduce_Request request ) [inline], [private]
```

Given GRPC Reduce request , Given output file , runs user defined reduce function and emits output data.

Parameters

<i>request</i>	grpc Reduce Request , check .proto
----------------	------------------------------------

Returns

grpc Reduce Response payload , check .proto

Definition at line 393 of file worker.h.

```
394 {
```



```

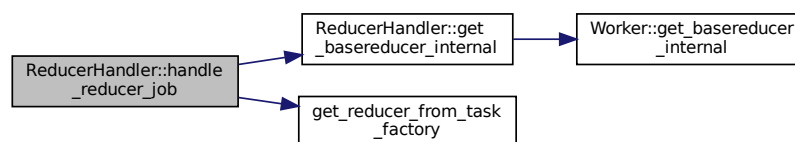
395     masterworker::Reduce_Response payload;
396     auto user_reducer_func = get_reducer_from_task_factory(request.uuid());
397     auto base_reducer = get_basereducer_internal(user_reducer_func.get());
398     base_reducer->file_name = request.output_file();
399     std::map<std::string, std::vector<std::string>> key_value_map;
400     payload.set_file_name(request.output_file());
401     auto d = request.file_list();
402     for (const auto& f : d)
403     {
404         std::ifstream fs(f);
405         std::string dummy;
406         try{
407             if (fs.good() && fs.is_open())
408             {
409                 while (std::getline(fs, dummy))
410                 {
411                     key_value_map[dummy.substr(0, dummy.find_first_of(DELIMITER))].push_back(
412                         dummy.substr(dummy.find_first_of(DELIMITER) + 1));
413                 }
414             }
415         } catch (std::ifstream::failure &e) {
416             std::cerr << f << " Error: " << e.what() << std::endl;
417         }
418     }
419     for (const auto& k : key_value_map)
420     {
421         user_reducer_func->reduce(k.first, k.second);
422     }
423     key_value_map.clear();
424     return payload;
425 }
426 }

```

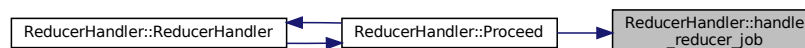
References DELIMITER, get_basereducer_internal(), and get_reducer_from_task_factory().

Referenced by Proceed().

Here is the call graph for this function:



Here is the caller graph for this function:



6.15.3.3 Proceed()

```
void ReducerHandler::Proceed ( ) [inline], [virtual]
```

Reimplemented from [BaseHandler](#).

Definition at line 135 of file worker.h.

```

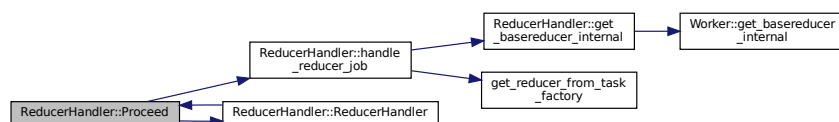
136     {
137         if (status_ == CREATE)
138         {
139             status_ = PROCESS;
140             service->Requestreduce(&ctx_, &ReduceRequest, &r_writer, s_queue, s_queue, this);
141         }
142         else if (status_ == PROCESS)
143         {
144             new ReducerHandler(service, s_queue, worker_address);
145             ReduceResponse = handle_reducer_job(ReduceRequest);
146             status_ = FINISH;
147             r_writer.Finish(ReduceResponse, grpc::Status::OK, this);
148         }
149         else
150         {
151             GPR_ASSERT(status_ == FINISH);
152             delete this;
153         }
154     }

```

References [BaseHandler::CREATE](#), [BaseHandler::ctx_](#), [BaseHandler::FINISH](#), [handle_reducer_job\(\)](#), [BaseHandler::PROCESS](#), [r_writer](#), [ReduceRequest](#), [ReduceResponse](#), [ReducerHandler\(\)](#), [BaseHandler::s_queue](#), [BaseHandler::service](#), [BaseHandler::status_](#), and [BaseHandler::worker_address](#).

Referenced by [ReducerHandler\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.15.4 Field Documentation

6.15.4.1 r_writer

```
grpc::ServerAsyncResponseWriter<masterworker::Reduce_Response> ReducerHandler::r_writer [private]
```

Definition at line 159 of file worker.h.

Referenced by [Proceed\(\)](#).

6.15.4.2 ReduceRequest

```
masterworker::Reduce_Request ReducerHandler::ReduceRequest [private]
```

Definition at line 157 of file worker.h.

Referenced by Proceed().

6.15.4.3 ReduceResponse

```
masterworker::Reduce_Response ReducerHandler::ReduceResponse [private]
```

Definition at line 158 of file worker.h.

Referenced by Proceed().

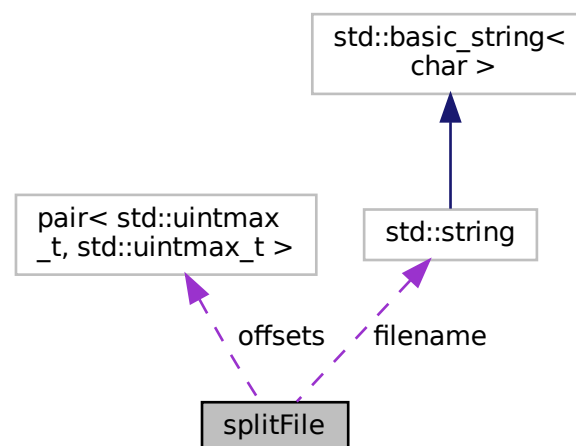
The documentation for this class was generated from the following file:

- [src/worker.h](#)

6.16 splitFile Struct Reference

```
#include <file_shard.h>
```

Collaboration diagram for splitFile:



Data Fields

- `std::string` [filename](#)
- `std::pair< std::uintmax_t, std::uintmax_t >` [offsets](#)

6.16.1 Detailed Description

Definition at line 25 of file `file_shard.h`.

6.16.2 Field Documentation

6.16.2.1 filename

```
std::string splitFile::filename
```

Definition at line 27 of file `file_shard.h`.

Referenced by `MapperHandler::convert_grpc_spec()`, and `shard_files()`.

6.16.2.2 offsets

```
std::pair<std::uintmax_t, std::uintmax_t> splitFile::offsets
```

Definition at line 28 of file `file_shard.h`.

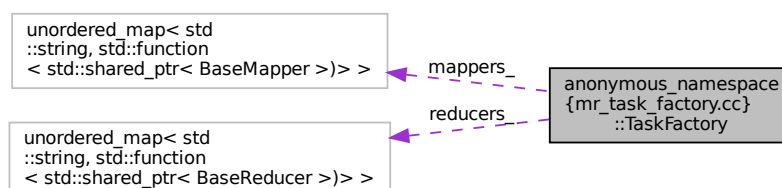
Referenced by `shard_files()`.

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

- `src/file_shard.h`

6.17 anonymous_namespace{mr_task_factory.cc}::TaskFactory Class Reference

Collaboration diagram for `anonymous_namespace{mr_task_factory.cc}::TaskFactory`:



Public Member Functions

- `std::shared_ptr< BaseMapper >` [get_mapper](#) (const `std::string` &`user_id`)
- `std::shared_ptr< BaseReducer >` [get_reducer](#) (const `std::string` &`user_id`)

Static Public Member Functions

- static [TaskFactory](#) & [instance](#) ()

Data Fields

- `std::unordered_map< std::string, std::function< std::shared_ptr< BaseMapper >> >` [mappers_](#)
- `std::unordered_map< std::string, std::function< std::shared_ptr< BaseReducer >> >` [reducers_](#)

Private Member Functions

- [TaskFactory](#) ()

6.17.1 Detailed Description

Definition at line 30 of file `mr_task_factory.cc`.

6.17.2 Constructor & Destructor Documentation

6.17.2.1 TaskFactory()

```
anonymous_namespace{mr_task_factory.cc}::TaskFactory::TaskFactory ( ) [private]
```

Definition at line 54 of file `mr_task_factory.cc`.

```
54 {}
```

6.17.3 Member Function Documentation

6.17.3.1 get_mapper()

```
std::shared_ptr< BaseMapper > anonymous_namespace{mr_task_factory.cc}::TaskFactory::get_mapper
(
    const std::string & user_id )
```

Definition at line 57 of file `mr_task_factory.cc`.

```
57
58     auto itr = mappers_.find(user_id);
59     if (itr == mappers_.end())
60         return nullptr;
61     return itr->second();
62 }
```

6.17.3.2 get_reducer()

```
std::shared_ptr< BaseReducer > anonymous_namespace{mr_task_factory.cc}::TaskFactory::get_reducer (
    const std::string & user_id )
```

Definition at line 65 of file mr_task_factory.cc.

```
65                                     {
66     auto itr = reducers_.find(user_id);
67     if (itr == reducers_.end())
68         return nullptr;
69     return itr->second();
70 }
```

6.17.3.3 instance()

```
TaskFactory & anonymous_namespace{mr_task_factory.cc}::TaskFactory::instance ( ) [static]
```

Definition at line 47 of file mr_task_factory.cc.

```
47                                     {
48
49     static TaskFactory *instance = new TaskFactory();
50     return *instance;
51 }
```

6.17.4 Field Documentation

6.17.4.1 mappers_

```
std::unordered_map<std::string, std::function<std::shared_ptr<BaseMapper>>> > anonymous_namespace{mr_task_factory.cc}::TaskFactory::mappers_
```

Definition at line 38 of file mr_task_factory.cc.

6.17.4.2 reducers_

```
std::unordered_map<std::string, std::function<std::shared_ptr<BaseReducer>>> > anonymous_namespace{mr_task_factory.cc}::TaskFactory::reducers_
```

Definition at line 39 of file mr_task_factory.cc.

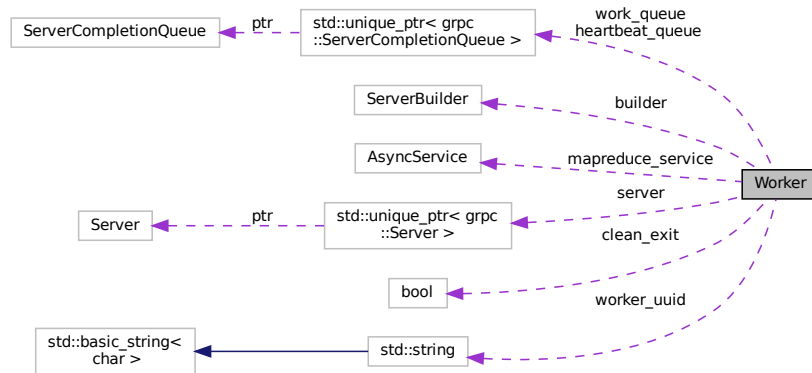
The documentation for this class was generated from the following file:

- [src/mr_task_factory.cc](#)

6.18 Worker Class Reference

```
#include <worker.h>
```

Collaboration diagram for Worker:



Public Member Functions

- [Worker](#) (std::string ip_addr_port)
- bool [run](#) ()
- [~Worker](#) ()

Static Public Member Functions

- static [BaseReducerInternal](#) * [get_basereducer_internal](#) (BaseReducer *reducer)
- static [BaseMapperInternal](#) * [get_basemapper_internal](#) (BaseMapper *mapper)

Private Member Functions

- void [heartbeat_handler](#) ()

Private Attributes

- grpc::ServerBuilder [builder](#)
- std::unique_ptr<grpc::ServerCompletionQueue> [work_queue](#)
- std::unique_ptr<grpc::ServerCompletionQueue> [heartbeat_queue](#)
- masterworker::Map_Reduce::AsyncService [mapreduce_service](#)
- std::unique_ptr<grpc::Server> [server](#)
- std::string [worker_uuid](#)
- bool [clean_exit](#) = false

6.18.1 Detailed Description

CS6210_TASK: Handle all the task a [Worker](#) is supposed to do. This is a big task for this project, will test your understanding of mapreduce

Definition at line 219 of file worker.h.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 Worker()

```
Worker::Worker (
    std::string ip_addr_port ) [inline]
```

ip_addr_port is the only information you get when started. You can populate your other class data members here if you want

Parameters

<i>ip_addr_port</i>	
---------------------	--

Definition at line 265 of file worker.h.

```
266 {
267     std::cout << "listening on " << ip_addr_port << std::endl;
268     Worker::builder.AddListeningPort(ip_addr_port, grpc::InsecureServerCredentials());
269     Worker::builder.RegisterService(&this->mapreduce_service);
270     Worker::work_queue = Worker::builder.AddCompletionQueue();
271     Worker::heartbeat_queue = Worker::builder.AddCompletionQueue();
272     Worker::worker_uuid = ip_addr_port.substr(ip_addr_port.find_first_of(':') + 1);
273 }
```

References builder, heartbeat_queue, mapreduce_service, work_queue, and worker_uuid.

6.18.2.2 ~Worker()

```
Worker::~Worker ( ) [inline]
```

Definition at line 229 of file worker.h.

```
230 {
231     Worker::clean_exit = true;
232     this->server->Shutdown();
233 }
```

References clean_exit, and server.

6.18.3 Member Function Documentation

6.18.3.1 get_basemapper_internal()

```
static BaseMapperInternal* Worker::get_basemapper_internal (
    BaseMapper * mapper ) [inline], [static]
```

Definition at line 240 of file worker.h.

```
241 {
242     return mapper->impl_;
243 }
```

Referenced by MapperHandler::get_basemapper_internal().

Here is the caller graph for this function:



6.18.3.2 get_basereducer_internal()

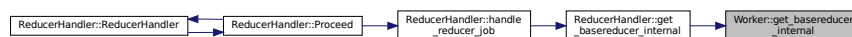
```
static BaseReducerInternal* Worker::get_basereducer_internal (
    BaseReducer * reducer ) [inline], [static]
```

Definition at line 235 of file worker.h.

```
236 {
237     return reducer->impl_;
238 }
```

Referenced by ReducerHandler::get_basereducer_internal().

Here is the caller graph for this function:



6.18.3.3 heartbeat_handler()

```
void Worker::heartbeat_handler ( ) [inline], [private]
```

Heartbeat handler , recives heartbeat request and send them back with same values and ALIVE state, unless clean_exit is marked true.

Definition at line 304 of file worker.h.

```
305 {
306     void* tag;
307     bool ok;
308
309     new HeartbeatHandler (&(Worker::mapreduce_service), heartbeat_queue.get(), worker_uuid);
310 }
```

```

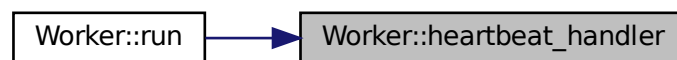
311     while (true)
312     {
313         if (Worker::clean_exit)
314             return;
315         GPR_ASSERT(heartbeat_queue->Next(&tag, &ok));
316
317         static_cast<BaseHandler*>(tag)->Proceed();
318     }
319 }

```

References `clean_exit`, `heartbeat_queue`, `mapreduce_service`, and `worker_uuid`.

Referenced by `run()`.

Here is the caller graph for this function:



6.18.3.4 run()

```
bool Worker::run ( ) [inline]
```

Here you go. once this function is called your woker's job is to keep looking for new tasks from [Master](#), complete when given one and again keep looking for the next one. Note that you have the access to `BaseMapper`'s member [BaseMapperInternal](#) `impl_` and `BaseReducer`'s member [BaseReducerInternal](#) `impl_` directly, so you can manipulate them however you want when running map/reduce tasks

Returns

Definition at line 284 of file `worker.h`.

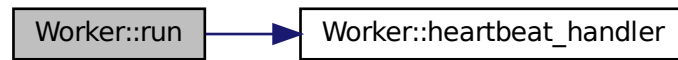
```

285 {
286     void* tag;
287     bool ok;
288     Worker::server = Worker::builder.BuildAndStart();
289     std::thread heartbeat_job(&Worker::heartbeat_handler, this);
290     new MapperHandler(&(Worker::mapreduce_service), work_queue.get(), worker_uuid);
291     new ReducerHandler(&(Worker::mapreduce_service), work_queue.get(), worker_uuid);
292
293     while (true)
294     {
295         GPR_ASSERT(work_queue->Next(&tag, &ok));
296         static_cast<BaseHandler*>(tag)->Proceed();
297     }
298     return true;
299 }

```

References `builder`, `heartbeat_handler()`, `mapreduce_service`, `server`, `work_queue`, and `worker_uuid`.

Here is the call graph for this function:



6.18.4 Field Documentation

6.18.4.1 builder

```
grpc::ServerBuilder Worker::builder [private]
```

Definition at line 248 of file worker.h.

Referenced by `run()`, and `Worker()`.

6.18.4.2 clean_exit

```
bool Worker::clean_exit = false [private]
```

Definition at line 257 of file worker.h.

Referenced by `heartbeat_handler()`, and `~Worker()`.

6.18.4.3 heartbeat_queue

```
std::unique_ptr<grpc::ServerCompletionQueue> Worker::heartbeat_queue [private]
```

Definition at line 250 of file worker.h.

Referenced by `heartbeat_handler()`, and `Worker()`.

6.18.4.4 mapreduce_service

masterworker::Map_Reduce::AsyncService Worker::mapreduce_service [private]

Definition at line 251 of file worker.h.

Referenced by heartbeat_handler(), run(), and Worker().

6.18.4.5 server

std::unique_ptr<grpc::Server> Worker::server [private]

Definition at line 252 of file worker.h.

Referenced by run(), and ~Worker().

6.18.4.6 work_queue

std::unique_ptr<grpc::ServerCompletionQueue> Worker::work_queue [private]

Definition at line 249 of file worker.h.

Referenced by run(), and Worker().

6.18.4.7 worker_uuid

std::string Worker::worker_uuid [private]

Definition at line 253 of file worker.h.

Referenced by heartbeat_handler(), run(), and Worker().

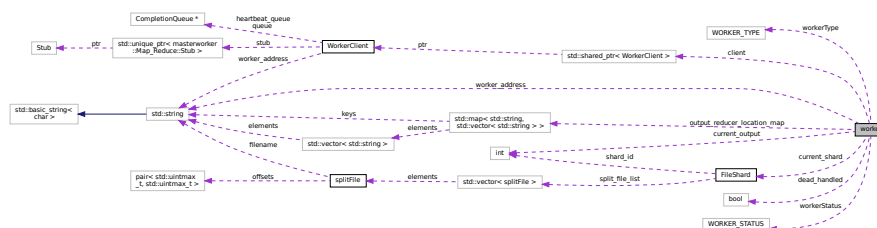
The documentation for this class was generated from the following file:

- [src/worker.h](#)

6.19 worker Struct Reference

```
#include <master.h>
```

Collaboration diagram for worker:



Data Fields

- `std::string` `worker_address`
- `WORKER_STATUS` `workerStatus`
- `WORKER_TYPE` `workerType`
- `FileShard` `current_shard`
- `std::shared_ptr< WorkerClient >` `client`
- `std::map< std::string, std::vector< std::string > >` `output_reducer_location_map`
- `int` `current_output`
- `bool` `dead_handled` = false

6.19.1 Detailed Description

Definition at line 238 of file master.h.

6.19.2 Field Documentation

6.19.2.1 client

```
std::shared_ptr<WorkerClient> worker::client
```

Definition at line 244 of file master.h.

Referenced by `Master::Master()`.

6.19.2.2 current_output

```
int worker::current_output
```

Definition at line 246 of file master.h.

6.19.2.3 current_shard

```
FileShard worker::current_shard
```

Definition at line 243 of file master.h.

6.19.2.4 `dead_handled`

```
bool worker::dead_handled = false
```

Definition at line 247 of file master.h.

6.19.2.5 `output_reducer_location_map`

```
std::map<std::string, std::vector<std::string> > worker::output_reducer_location_map
```

Definition at line 245 of file master.h.

6.19.2.6 `worker_address`

```
std::string worker::worker_address
```

Definition at line 240 of file master.h.

Referenced by `Master::async_map()`, `Master::async_reducer()`, and `Master::Master()`.

6.19.2.7 `workerStatus`

```
WORKER_STATUS worker::workerStatus
```

Definition at line 241 of file master.h.

Referenced by `Master::async_map()`, `Master::async_reducer()`, and `Master::Master()`.

6.19.2.8 `workerType`

```
WORKER_TYPE worker::workerType
```

Definition at line 242 of file master.h.

Referenced by `Master::Master()`.

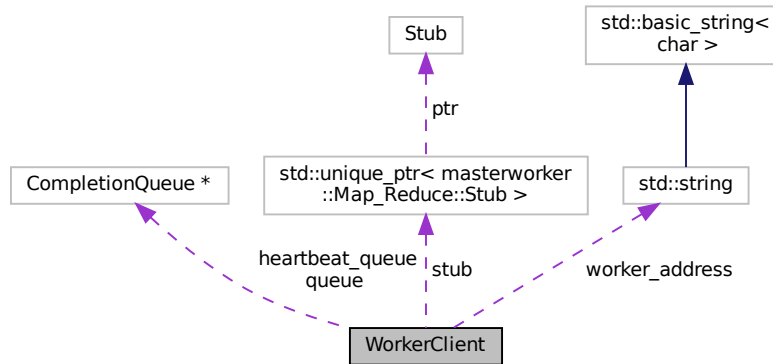
The documentation for this struct was generated from the following file:

- [src/master.h](#)

6.20 WorkerClient Class Reference

```
#include <master.h>
```

Collaboration diagram for WorkerClient:



Public Member Functions

- [WorkerClient](#) (const std::string &address, grpc::CompletionQueue *[queue](#))
- void [send_heartbeat](#) (int64_t current_time)
- bool [recv_heartbeat](#) ()
- void [schedule_reduce_job](#) (MapReduceSpec spec, std::vector< std::string > file_list, std::string output_↔ file_location)
- void [schedule_mapper_jobs](#) (MapReduceSpec spec, FileShard shard)
- ~[WorkerClient](#) ()

Private Member Functions

- void [convert_grpc_spec](#) (FileShard *shard, masterworker::partition *partition)

Private Attributes

- std::unique_ptr< masterworker::Map_Reduce::Stub > [stub](#)
- grpc::CompletionQueue * [queue](#)
- std::string [worker_address](#)
- grpc::CompletionQueue * [heartbeat_queue](#)

6.20.1 Detailed Description

[Worker](#) Client class to communicate with worker class

Definition at line 89 of file master.h.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 WorkerClient()

```
WorkerClient::WorkerClient (
    const std::string & address,
    grpc::CompletionQueue * queue )
```

Constructor for worker client , create communication insecure channel for each worker.

Parameters

<i>address</i>	
<i>queue</i>	

Definition at line 119 of file master.h.

```
120     : queue(queue)
121     , worker_address(address)
122 {
123     std::cout << "creating channel at " + address << std::endl;
124     heartbeat_queue = new grpc::CompletionQueue();
125     this->stub = masterworker::Map_Reduce::NewStub(grpc::CreateChannel(address,
126     grpc::InsecureChannelCredentials()));
126 }
```

References `heartbeat_queue`, and `stub`.

6.20.2.2 ~WorkerClient()

```
WorkerClient::~~WorkerClient ( ) [inline]
```

Definition at line 101 of file master.h.

```
102     {
103         heartbeat_queue->Shutdown();
104     }
```

References `heartbeat_queue`.

6.20.3 Member Function Documentation

6.20.3.1 convert_grpc_spec()

```
void WorkerClient::convert_grpc_spec (
    FileShard * shard,
    masterworker::partition * partition ) [private]
```

Convert `FileShard` struct to GRPC partition.

Parameters

<i>shard</i>	
<i>partition</i>	

Definition at line 206 of file master.h.

```

207 {
208     partition->set_shard_id(shard->shard_id);
209     for (auto f : shard->split_file_list)
210     {
211         auto temp = partition->add_file_list();
212         temp->set_filename(f.filename);
213         temp->set_start_offset(f.offsets.first);
214         temp->set_end_offset(f.offsets.second);
215     }
216 }
```

References FileShard::shard_id, and FileShard::split_file_list.

Referenced by schedule_mapper_jobs().

Here is the caller graph for this function:



6.20.3.2 recv_heartbeat()

```
bool WorkerClient::recv_heartbeat ( )
```

Async Heartbeat response reader ,

Returns

false if worker time out or unreachable or any other communication case.

Definition at line 153 of file master.h.

```

154 {
155     void* tag;
156     bool ok = false;
157     GPR_ASSERT(WorkerClient::heartbeat_queue->Next(&tag, &ok));
158     auto* call = static_cast<HeartbeatCall*>(tag);
159     if (call->status.ok())
160     {
161         if (call->result.status() == masterworker::Heartbeat_Payload_type_DEAD)
162         {
163             std::cerr << "Error " << call->worker_ip_addr << " : Dead" << std::endl;
164             return false;
165         }
166         delete call;
167         return true;
168     }
169     auto temp = call->status;
170     std::cerr << "Error " << this->worker_address << " : " << call->status.error_message()
171     << "details : " << call->status.error_details() << "status code " << call->status.error_code()
172     << " ok "
173     << call->status.ok() << std::endl;
174     return false;
175 }
```

References heartbeat_queue, worker_address, and AsyncClientCall::worker_ip_addr.

6.20.3.3 schedule_mapper_jobs()

```
void WorkerClient::schedule_mapper_jobs (
    MapReduceSpec spec,
    FileShard shard )
```

Schedules mapper job with [Worker](#) Client, similar to reduce , heartbeat etc

Parameters

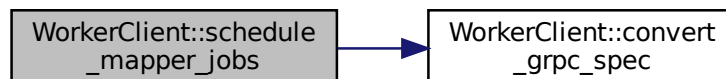
<i>spec</i>	
<i>shard</i>	

Definition at line 223 of file master.h.

```
224 {
225     masterworker::Map_Request mapRequest;
226     mapRequest.set_uuid(spec.user);
227     mapRequest.set_partition_count(spec.output_files);
228     auto s = mapRequest.add_shard();
229     this->convert_grpc_spec(&shard, s);
230     auto call = new MapCall;
231     call->worker_ip_addr = WorkerClient::worker_address;
232     call->map_response_reader = WorkerClient::stub->PrepareAsyncmap(&call->context, mapRequest,
WorkerClient::queue);
233     call->is_map_job = true;
234     call->map_response_reader->StartCall();
235     call->map_response_reader->Finish(&call->result, &call->status, (void*)call);
236 }
```

References [convert_grpc_spec\(\)](#), [MapReduceSpec::output_files](#), [queue](#), [stub](#), [MapReduceSpec::user](#), [worker_address](#), and [AsyncClientCall::worker_ip_addr](#).

Here is the call graph for this function:



6.20.3.4 schedule_reduce_job()

```
void WorkerClient::schedule_reduce_job (
    MapReduceSpec spec,
    std::vector< std::string > file_list,
    std::string output_file_location )
```

Schedules Reduce Jobs to worker.

Parameters

<i>spec</i>	mapreduce ini file data structure.
<i>file_list</i>	
<i>output_file_location</i>	

Definition at line 181 of file master.h.

```

185 {
186     masterworker::Reduce_Request reduceRequest;
187     reduceRequest.set_uuid(spec.user);
188     reduceRequest.set_output_file(output_file_location);
189     for (const auto& l : file_list)
190     {
191         auto f = reduceRequest.add_file_list();
192         f->append(l);
193     }
194     auto call = new ReduceCall;
195     call->worker_ip_addr = this->worker_address;
196     call->reducer_response_reader = WorkerClient::stub->PrepareAsyncreduce(&call->context,
197     reduceRequest, WorkerClient::queue);
197     call->is_map_job = false;
198     call->reducer_response_reader->StartCall();
199     call->reducer_response_reader->Finish(&call->result, &call->status, (void*)call);
200 }
```

References `queue`, `stub`, `MapReduceSpec::user`, `worker_address`, and `AsyncClientCall::worker_ip_addr`.

6.20.3.5 send_heartbeat()

```

void WorkerClient::send_heartbeat (
    int64_t current_time )
```

Send Async Heartbeat Request to client to infer health. It sets deadline for TIMEOUT 5 secs if worker times out in any case

Parameters

<i>current_time</i>	
---------------------	--

Definition at line 132 of file master.h.

```

133 {
134     //      std::cout << "Info " << std::chrono::system_clock::to_time_t(std::chrono::system_clock::now()) <<
135     //      " " +
136     //      this->worker_address << " : Hbeat sent" << std::endl;
137     std::chrono::system_clock::time_point deadline = std::chrono::system_clock::now() +
138     std::chrono::seconds(TIMEOUT);
139     auto call = new HeartbeatCall;
140     call->worker_ip_addr = this->worker_address;
141     call->context.set_deadline(deadline);
142     masterworker::Heartbeat_Payload payload;
143     payload.set_id(this->worker_address);
144     //      payload.set_timestamp(current_time);
145     payload.set_status(masterworker::Heartbeat_Payload_type_UNKNOWN);
146     call->heartbeat_payload_reader = WorkerClient::stub->PrepareAsyncheartbeat(&call->context, payload,
147     WorkerClient::heartbeat_queue);
148     call->heartbeat_payload_reader->StartCall();
149     call->heartbeat_payload_reader->Finish(&call->result, &call->status, (void*)call);
150 }
```

References `heartbeat_queue`, `stub`, `TIMEOUT`, `worker_address`, and `AsyncClientCall::worker_ip_addr`.

6.20.4 Field Documentation

6.20.4.1 heartbeat_queue

```
grpc::CompletionQueue* WorkerClient::heartbeat_queue [private]
```

Definition at line 110 of file master.h.

Referenced by `recv_heartbeat()`, `send_heartbeat()`, `WorkerClient()`, and `~WorkerClient()`.

6.20.4.2 queue

```
grpc::CompletionQueue* WorkerClient::queue [private]
```

Definition at line 108 of file master.h.

Referenced by `schedule_mapper_jobs()`, and `schedule_reduce_job()`.

6.20.4.3 stub

```
std::unique_ptr<masterworker::Map_Reduce::Stub> WorkerClient::stub [private]
```

Definition at line 107 of file master.h.

Referenced by `schedule_mapper_jobs()`, `schedule_reduce_job()`, `send_heartbeat()`, and `WorkerClient()`.

6.20.4.4 worker_address

```
std::string WorkerClient::worker_address [private]
```

Definition at line 109 of file master.h.

Referenced by `recv_heartbeat()`, `schedule_mapper_jobs()`, `schedule_reduce_job()`, and `send_heartbeat()`.

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

- [src/master.h](#)

Chapter 7

File Documentation

7.1 src/CMakeLists.txt File Reference

Functions

- [cmake_minimum_required](#) (VERSION 3.10) project(project4) set(CMAKE_CXX_STANDARD 17) include(Generate↵ Protos.cmake) add_library(mapreducelib mapreduce.cc mapreduce_impl.cc master.h mapreduce_↵ spec.h file_shard.h) if(CMAKE_COMPILER_IS_GNUCC AND CMAKE_CXX_COMPILER_VERSION VERSION_GREATER 7) target_link_libraries(mapreducelib p4protolib stdc++fs) else() target_link_↵ libraries(mapreducelib p4protolib) endif() target_include_directories(mapreducelib PUBLIC \$
- [add_dependencies](#) (mapreducelib p4protolib) add_library(mr_workerlib mr_task_factory.cc run_worker.cc mr_tasks.h worker.h) if(CMAKE_COMPILER_IS_GNUCC AND CMAKE_CXX_COMPILER_VERSION VE↵ RSION_GREATER 7) target_link_libraries(mr_workerlib p4protolib stdc++fs) else() target_link_libraries(mr_↵ _workerlib p4protolib) endif() target_include_directories(mr_workerlib PUBLIC \$

7.1.1 Function Documentation

7.1.1.1 add_dependencies()

```
add_dependencies (
    mapreducelib p4protolib )
```

Definition at line 20 of file CMakeLists.txt.

33

{MAPREDUCE_INCLUDE_DIR})

7.1.1.2 cmake_minimum_required()

```
cmake_minimum_required (
    VERSION 3. 10 )
```

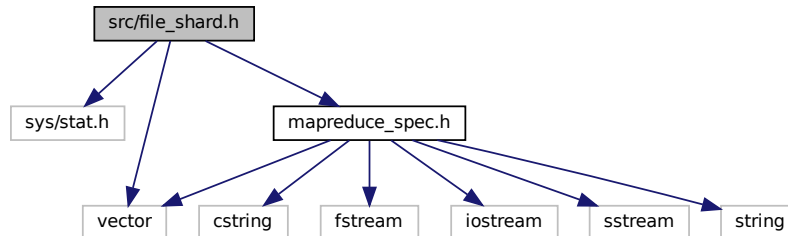
Definition at line 2 of file CMakeLists.txt.

19

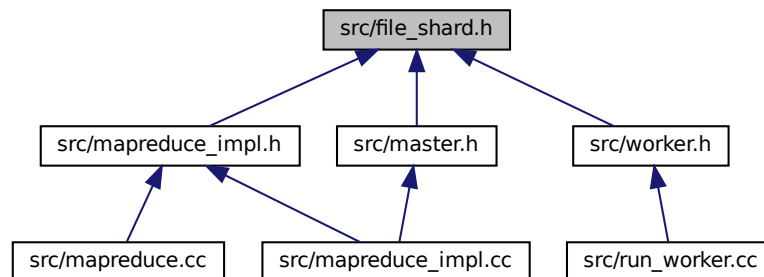
{MAPREDUCE_INCLUDE_DIR})

7.2 src/file_shard.h File Reference

```
#include <sys/stat.h>
#include <vector>
#include "mapreduce_spec.h"
Include dependency graph for file_shard.h:
```



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



Data Structures

- struct [splitFile](#)
- struct [FileShard](#)

Macros

- `#define` [KB](#) 1024
- `#define` [TEMP_DIR](#) "intermediate"

Functions

- `std::uintmax_t` [get_filesize](#) (std::string path)
- `std::uintmax_t` [approx_split](#) (const std::basic_string< char > fileName, uintmax_t offset, uintmax_t optimal←_shard_size)
- `bool` [shard_files](#) (const [MapReduceSpec](#) &mr_spec, std::vector< [FileShard](#) > &fileShards)

7.2.1 Macro Definition Documentation

7.2.1.1 KB

```
#define KB 1024
```

Definition at line 7 of file file_shard.h.

7.2.1.2 TEMP_DIR

```
#define TEMP_DIR "intermediate"
```

Definition at line 8 of file file_shard.h.

7.2.2 Function Documentation

7.2.2.1 approx_split()

```
std::uintmax_t approx_split (
    const std::basic_string< char > fileName,
    uintmax_t offset,
    uintmax_t optimal_shard_size ) [inline]
```

finds nearest
location in given shard file. usually its after the optimal size.

Parameters

<i>fileName</i>	
<i>offset</i>	
<i>optimal_shard_size</i>	

Returns

nearest file Offset for \n from given file offset. usually used as offset + return_value

Definition at line 46 of file file_shard.h.

```
50 {
51     std::uintmax_t approx_size;
52     std::ifstream fs(fileName);
53     if (!fs.good())
54     {
55         std::cerr << "Error Opening file: " << fileName << std::endl;
```

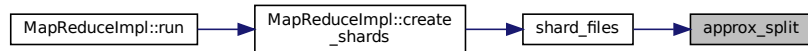
```

56         return 0;
57     }
58     fs.seekg(offset + optimal_shard_size);
59     std::string temp_str;
60     std::getline(fs, temp_str);
61     approx_size = optimal_shard_size + temp_str.length() + 1;
62     return approx_size;
63 }

```

Referenced by `shard_files()`.

Here is the caller graph for this function:



7.2.2.2 get_filesize()

```

std::uintmax_t get_filesize (
    std::string path ) [inline]

```

Boiler plate function for retrieving file size

Parameters

<i>path</i>	
-------------	--

Returns

file size usually with 64bit value

Definition at line 14 of file `file_shard.h`.

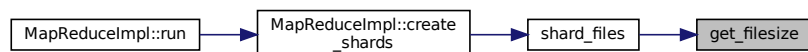
```

15 {
16 #if __cplusplus >= 201703L
17     return fs::file_size(path);
18 #else
19     struct stat stat_buf;
20     int rc = stat(path.c_str(), &stat_buf);
21     return rc == 0 ? stat_buf.st_size : -1;
22 #endif
23 }

```

Referenced by `shard_files()`.

Here is the caller graph for this function:



7.2.2.3 shard_files()

```
bool shard_files (
    const MapReduceSpec & mr_spec,
    std::vector< FileShard > & fileShards ) [inline]
```

Create file shards from the list of input files, map_kilobytes * etc. using mr_spec you populated

Parameters

<i>mr_spec</i>	
<i>fileShards</i>	

Returns

true if succeeded.

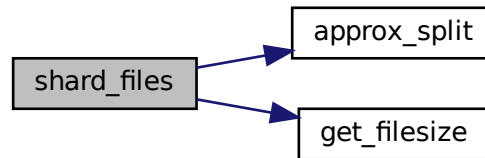
Definition at line 72 of file file_shard.h.

```
73 {
74     std::uintmax_t optimal_shard_size = mr_spec.map_kb * KB;
75     std::intmax_t rem_shard_size = optimal_shard_size;
76     FileShard current_shard;
77     current_shard.shard_id = fileShards.size();
78     for (const auto& f : mr_spec.input_files)
79     {
80         std::uintmax_t file_size, rem_file_size;
81         file_size = rem_file_size = get_filesize(f);
82         std::uintmax_t offset = 0;
83         splitFile current_split_file;
84         while (rem_file_size > 0)
85         {
86             current_split_file.filename = f;
87             if (rem_shard_size >= rem_file_size)
88             {
89                 current_split_file.offsets = {offset, offset + rem_file_size};
90                 rem_shard_size -= rem_file_size;
91                 rem_file_size = 0;
92                 current_shard.split_file_list.push_back(current_split_file);
93             }
94             else
95             {
96                 std::uintmax_t nearest_size;
97                 nearest_size = offset + optimal_shard_size > file_size ? file_size - offset
98                                     : approx_split(f, offset,
99 rem_shard_size);
100                 current_split_file.offsets = {offset, offset + nearest_size};
101                 if (offset > offset + nearest_size)
102                 {
103                     perror("SOMETHING WENT WRONG.....");
104                     exit(1);
105                 }
106                 current_shard.split_file_list.push_back(current_split_file);
107                 current_split_file = splitFile();
108                 rem_shard_size -= nearest_size;
109                 rem_file_size -= nearest_size;
110                 offset += nearest_size;
111             }
112             if (rem_shard_size <= 0)
113             {
114                 fileShards.push_back(current_shard);
115                 current_shard = FileShard();
116                 current_shard.shard_id = fileShards.size();
117                 rem_shard_size = optimal_shard_size;
118             }
119         }
120         if (current_shard.shard_id > -1)
121             fileShards.push_back(current_shard);
122         return true;
123 }
```

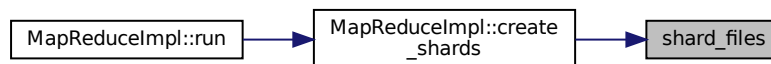
References [approx_split\(\)](#), [splitFile::filename](#), [get_filesize\(\)](#), [MapReduceSpec::input_files](#), [KB](#), [MapReduceSpec::map_kb](#), [splitFile::offsets](#), [FileShard::shard_id](#), and [FileShard::split_file_list](#).

Referenced by MapReduceImpl::create_shards().

Here is the call graph for this function:

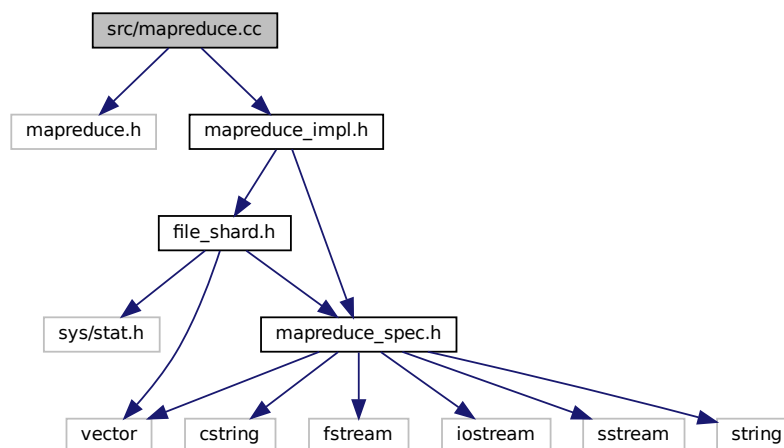


Here is the caller graph for this function:



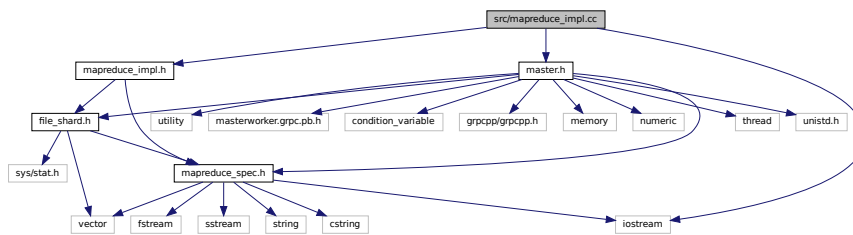
7.3 src/mapreduce.cc File Reference

```
#include <mapreduce.h>
#include "mapreduce_impl.h"
Include dependency graph for mapreduce.cc:
```



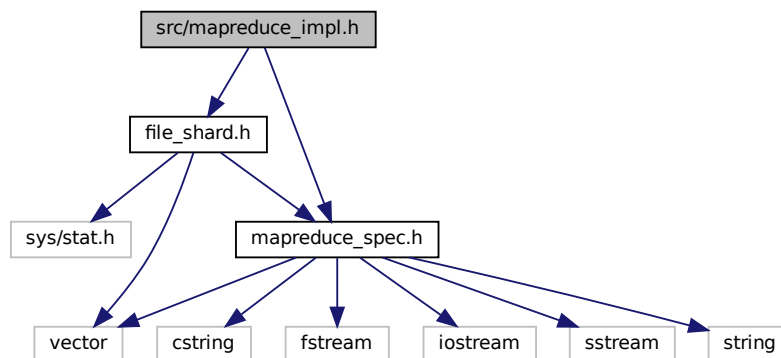
7.4 src/mapreduce_impl.cc File Reference

```
#include "mapreduce_impl.h"
#include <iostream>
#include "master.h"
Include dependency graph for mapreduce_impl.cc:
```

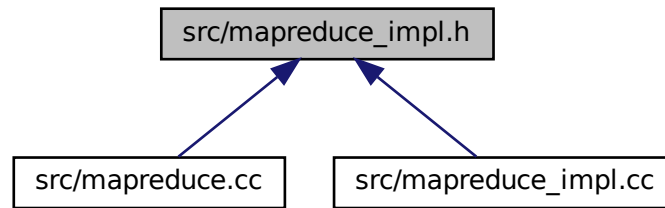


7.5 src/mapreduce_impl.h File Reference

```
#include "file_shard.h"
#include "mapreduce_spec.h"
Include dependency graph for mapreduce_impl.h:
```



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



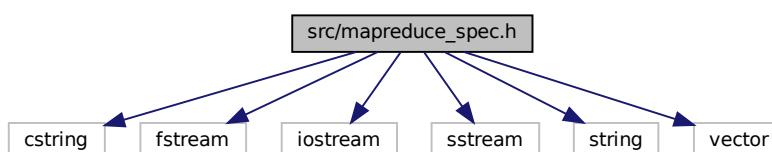
Data Structures

- class [MapReduceImpl](#)

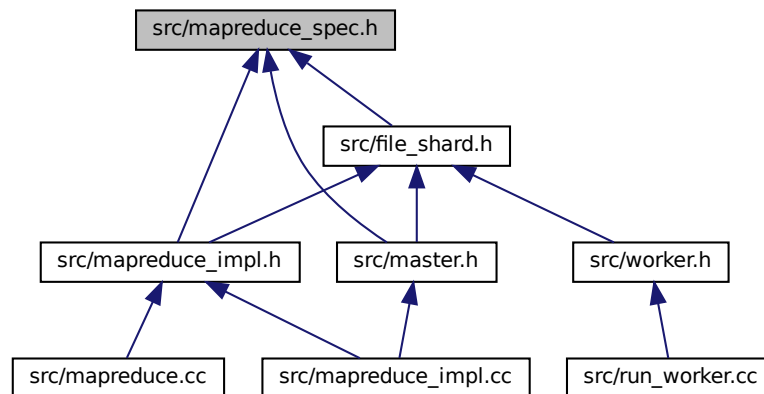
7.6 src/mapreduce_spec.h File Reference

```
#include <cstring>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
```

Include dependency graph for `mapreduce_spec.h`:



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



Data Structures

- struct [MapReduceSpec](#)

Functions

- `std::vector< std::string > splitString (const std::string &s, char del)`
- `bool read_mr_spec_from_config_file (const std::string &config_filename, MapReduceSpec &mr_spec)`
- `bool validate_mr_spec (const MapReduceSpec &mr_spec)`

7.6.1 Function Documentation

7.6.1.1 `read_mr_spec_from_config_file()`

```

bool read_mr_spec_from_config_file (
    const std::string & config_filename,
    MapReduceSpec & mr_spec ) [inline]

```

Populate [MapReduceSpec](#) data structure with the specification from the config file

Parameters

<i>config_filename</i>	
<i>mr_spec</i>	

Returns

true or false based on success

Definition at line 59 of file mapreduce_spec.h.

```

60 {
61     std::ifstream config_file(config_filename);
62     std::string config_line;
63     if (!config_file.good())
64     {
65         std::cerr << "Error opening file : " << config_filename << " Error No" << std::strerror(errno) <<
        std::endl;
66         return false;
67     }
68     while (std::getline(config_file, config_line))
69     {
70         std::string key, value;
71         key = config_line.substr(0, config_line.find_first_of('='));
72         value = config_line.substr(config_line.find_first_of('=') + 1, config_line.length());
73         if (value.empty() || key.empty())
74         {
75             std::cerr << key << " is empty or value " << value << " is empty , please check again" <<
            std::endl;
76             return false;
77         }
78         if (key == "n_workers")
79         {
80             mr_spec.worker_count = std::stoi(value);
81             continue;
82         }
83         if (key == "worker_ipaddr_ports")
84         {
85             mr_spec.worker_endpoints = splitString(value, ',');
86             continue;
87         }
88         if (key == "input_files")
89         {
90             mr_spec.input_files = splitString(value, ',');
91             continue;
92         }
93         if (key == "output_dir")
94         {
95             mr_spec.output_directory = value;
96             continue;
97         }
98         if (key == "n_output_files")
99         {
100             mr_spec.output_files = std::stoi(value);
101             continue;
102         }
103         if (key == "map_kilobytes")
104         {
105             mr_spec.map_kb = std::stoi(value);
106             continue;
107         }
108         if (key == "user_id")
109         {
110             mr_spec.user = value;
111             continue;
112         }
113     }
114 }
115
116 return true;
117 }
```

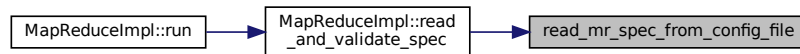
References `MapReduceSpec::input_files`, `MapReduceSpec::map_kb`, `MapReduceSpec::output_directory`, `MapReduceSpec::output_files`, `splitString()`, `MapReduceSpec::user`, `MapReduceSpec::worker_count`, and `MapReduceSpec::worker_endpoints`.

Referenced by `MapReduceImpl::read_and_validate_spec()`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.1.2 splitString()

```
std::vector<std::string> splitString (
    const std::string & s,
    char del ) [inline]
```

Splits string with given delimiter `del`

Parameters

<i>s</i>	--> raw string
<i>del</i>	--> delimiter

Returns

returns vector with parsed strings.

Definition at line 40 of file `mapreduce_spec.h`.

```
41 {
42     std::vector<std::string> arr{};
43     std::stringstream ss(s);
44     std::string temp;
45     while (std::getline(ss, temp, del))
46     {
47         arr.push_back(temp);
48     }
49     return arr;
50 }
```

Referenced by `read_mr_spec_from_config_file()`.

Here is the caller graph for this function:



7.6.1.3 validate_mr_spec()

```
bool validate_mr_spec (
    const MapReduceSpec & mr_spec ) [inline]
```

validate the specification read from the config file

Parameters

<i>mr_spec</i>	
----------------	--

Returns

true or false based on validation criteria.

Definition at line 125 of file mapreduce_spec.h.

```

126 {
127
128     if (mr_spec.worker_endpoints.size() != mr_spec.worker_count)
129     {
130         std::cerr << "Invalid Count of Workers : " << mr_spec.worker_endpoints.size() << "config -
worker_count"
131             << mr_spec.worker_count << std::endl;
132         return false;
133     }
134
135     #if __cplusplus >= 201703L
136     if (!fs::is_directory(mr_spec.output_directory))
137     {
138         if (fs::is_regular_file(mr_spec.output_directory))
139         {
140             std::cerr << mr_spec.output_directory << " is file not directory please provide correct path."
<< std::endl;
141             return false;
142         }
143         else
144         {
145             try
146             {
147                 fs::create_directory(mr_spec.output_directory);
148             }
149             catch (fs::filesystem_error& e)
150             {
151                 std::cout << e.what() << std::endl;
152             }
153         }
154     }
155     #endif
156     for (const auto& f : mr_spec.input_files)
157     {
158         std::ifstream temp_stream(f);
159         if (!temp_stream.good())
160         {
161             std::cerr << "Error opening fie : " << f << " Error No: " << std::strerror(errno) << std::endl;
162             return false;
163         }
164     }
165 }
```



```

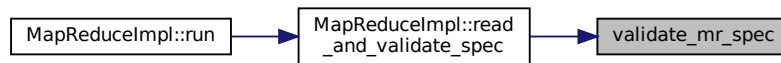
164     }
165     return true;
166 }

```

References `MapReduceSpec::input_files`, `MapReduceSpec::output_directory`, `MapReduceSpec::worker_count`, and `MapReduceSpec::worker_endpoints`.

Referenced by `MapReduceImpl::read_and_validate_spec()`.

Here is the caller graph for this function:



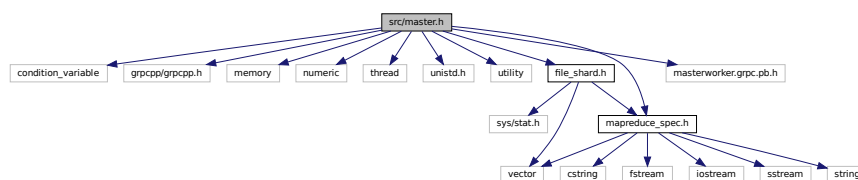
7.7 src/master.h File Reference

```

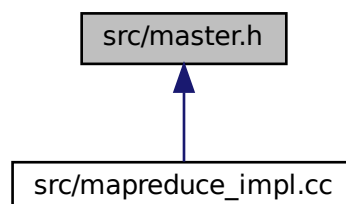
#include <condition_variable>
#include <grpcpp/grpcpp.h>
#include <memory>
#include <numeric>
#include <thread>
#include <unistd.h>
#include <utility>
#include "file_shard.h"
#include "mapreduce_spec.h"
#include "masterworker.grpc.pb.h"

```

Include dependency graph for master.h:



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



Data Structures

- struct [heartbeat_payload](#)
- class [AsyncClientCall](#)
- class [MapCall](#)
- class [ReduceCall](#)
- class [HeartbeatCall](#)
- class [WorkerClient](#)
- struct [worker](#)
- class [Master](#)

Macros

- `#define ALIVE true`
- `#define TIMEOUT 5`

Enumerations

- enum [WORKER_STATUS](#) { [FREE](#), [BUSY](#), [DEAD](#) }
- enum [WORKER_TYPE](#) { [MAPPER](#), [REDUCER](#) }

7.7.1 Macro Definition Documentation

7.7.1.1 [ALIVE](#)

```
#define ALIVE true
```

Definition at line 22 of file master.h.

7.7.1.2 [TIMEOUT](#)

```
#define TIMEOUT 5
```

Definition at line 24 of file master.h.

7.7.2 Enumeration Type Documentation

7.7.2.1 [WORKER_STATUS](#)

```
enum WORKER\_STATUS
```

Enumerator

FREE	
BUSY	
DEAD	

Definition at line 26 of file master.h.

```

27 {
28     FREE,
29     BUSY,
30     DEAD
31 };

```

7.7.2.2 WORKER_TYPE

```
enum WORKER_TYPE
```

Enumerator

MAPPER	
REDUCER	

Definition at line 32 of file master.h.

```

33 {
34     MAPPER,
35     REDUCER
36 };

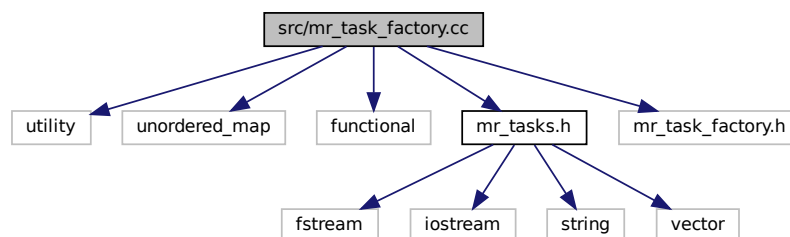
```

7.8 src/mr_task_factory.cc File Reference

```

#include <utility>
#include <unordered_map>
#include <functional>
#include "mr_tasks.h"
#include <mr_task_factory.h>
Include dependency graph for mr_task_factory.cc:

```



Data Structures

- class [anonymous_namespace{mr_task_factory.cc}::TaskFactory](#)

Namespaces

- [anonymous_namespace{mr_task_factory.cc}](#)

Functions

- bool [register_tasks](#) (std::string user_id, std::function< std::shared_ptr< BaseMapper >() > &generate_mapper, std::function< std::shared_ptr< BaseReducer >() > &generate_reducer)
- std::shared_ptr< BaseMapper > [get_mapper_from_task_factory](#) (const std::string &user_id)
- std::shared_ptr< BaseReducer > [get_reducer_from_task_factory](#) (const std::string &user_id)

7.8.1 Function Documentation

7.8.1.1 get_mapper_from_task_factory()

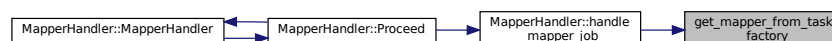
```
std::shared_ptr<BaseMapper> get_mapper_from_task_factory (
    const std::string & user_id )
```

Definition at line 81 of file mr_task_factory.cc.

```
81                                     {
82     return TaskFactory::instance() .get_mapper(user_id);
83 }
```

Referenced by MapperHandler::handle_mapper_job().

Here is the caller graph for this function:



7.8.1.2 get_reducer_from_task_factory()

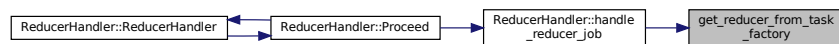
```
std::shared_ptr<BaseReducer> get_reducer_from_task_factory (
    const std::string & user_id )
```

Definition at line 86 of file mr_task_factory.cc.

```
86                                     {
87     return TaskFactory::instance().get_reducer(user_id);
88 }
```

Referenced by ReducerHandler::handle_reducer_job().

Here is the caller graph for this function:



7.8.1.3 register_tasks()

```
bool register_tasks (
    std::string user_id,
    std::function< std::shared_ptr< BaseMapper >() > & generate_mapper,
    std::function< std::shared_ptr< BaseReducer >() > & generate_reducer )
```

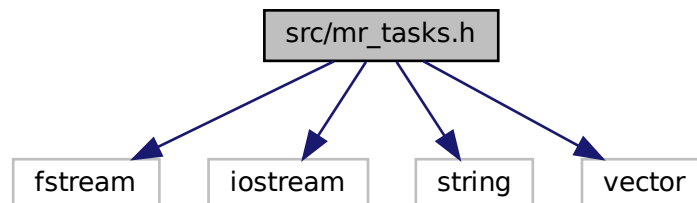
Definition at line 74 of file mr_task_factory.cc.

```
75                                     {
76     TaskFactory& factory = TaskFactory::instance();
77     return factory.mappers_.insert(std::make_pair(user_id, generate_mapper)).second
78     && factory.reducers_.insert(std::make_pair(user_id, generate_reducer)).second;
79 }
```

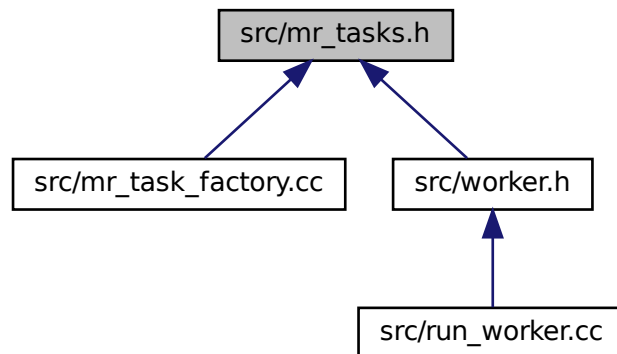
7.9 src/mr_tasks.h File Reference

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
```

Include dependency graph for mr_tasks.h:



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



Data Structures

- struct [BaseMapperInternal](#)
- struct [BaseReducerInternal](#)

Macros

- #define [DEBUG](#) 0
- #define [devnull](#) "/dev/null"
- #define [MAX_KV_PAIR_SIZE](#) 4096
- #define [DELIMITER](#) '|'

7.9.1 Macro Definition Documentation

7.9.1.1 DEBUG

```
#define DEBUG 0
```

Definition at line 8 of file `mr_tasks.h`.

7.9.1.2 DELIMITER

```
#define DELIMITER '|'
```

Definition at line 11 of file `mr_tasks.h`.

7.9.1.3 devnull

```
#define devnull "/dev/null"
```

Definition at line 9 of file mr_tasks.h.

7.9.1.4 MAX_KV_PAIR_SIZE

```
#define MAX_KV_PAIR_SIZE 4096
```

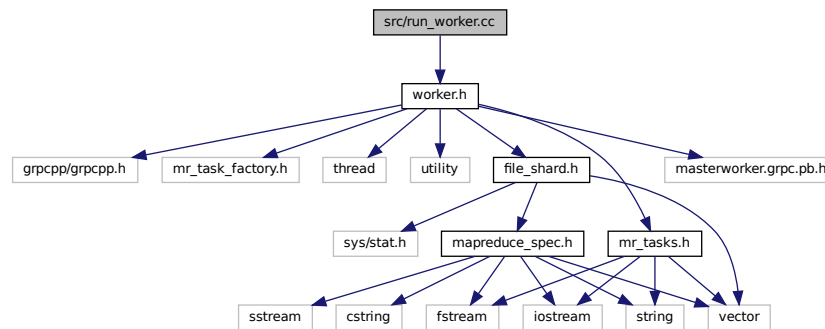
Definition at line 10 of file mr_tasks.h.

7.10 src/Readme.txt File Reference

7.11 src/run_worker.cc File Reference

```
#include "worker.h"
```

Include dependency graph for run_worker.cc:



Functions

- int [main](#) (int argc, char **argv)

7.11.1 Function Documentation

7.11.1.1 main()

```
int main (
    int argc,
    char ** argv )
```

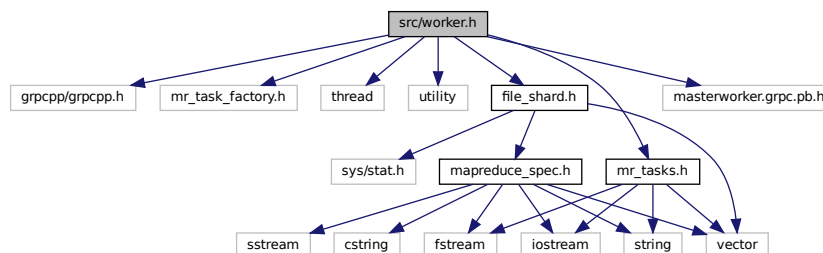
Definition at line 3 of file run_worker.cc.

```
4 {
5     std::string ip_addr_port;
6     if (argc == 2)
7     {
8         ip_addr_port = std::string(argv[1]);
9     }
10    else
11    {
12        std::cerr << "Correct usage: [$binary_name $ip_addr_port], example: [./mr_worker localhost:50051]"
13        << std::endl;
14        return EXIT_FAILURE;
15    }
16    Worker worker(ip_addr_port);
17    return worker.run() ? EXIT_SUCCESS : EXIT_FAILURE;
18 }
```

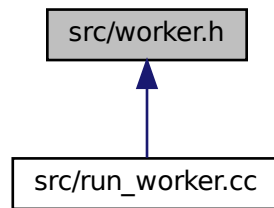
7.12 src/worker.h File Reference

```
#include <grpcpp/grpcpp.h>
#include <mr_task_factory.h>
#include <thread>
#include <utility>
#include "file_shard.h"
#include "masterworker.grpc.pb.h"
#include "mr_tasks.h"
```

Include dependency graph for worker.h:



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



Data Structures

- class [BaseHandler](#)
- class [MapperHandler](#)
- class [ReducerHandler](#)
- class [HeartbeatHandler](#)
- class [Worker](#)

Functions

- `std::shared_ptr< BaseMapper > get_mapper_from_task_factory (const std::string &user_id)`
- `std::shared_ptr< BaseReducer > get_reducer_from_task_factory (const std::string &user_id)`

7.12.1 Function Documentation

7.12.1.1 `get_mapper_from_task_factory()`

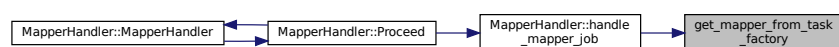
```
std::shared_ptr<BaseMapper> get_mapper_from_task_factory (
    const std::string & user_id )
```

Definition at line 81 of file `mr_task_factory.cc`.

```
81                                     {
82     return TaskFactory::instance().get_mapper(user_id);
83 }
```

Referenced by `MapperHandler::handle_mapper_job()`.

Here is the caller graph for this function:



7.12.1.2 get_reducer_from_task_factory()

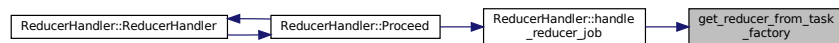
```
std::shared_ptr<BaseReducer> get_reducer_from_task_factory (
    const std::string & user_id )
```

Definition at line 86 of file mr_task_factory.cc.

```
86                                     {
87     return TaskFactory::instance().get_reducer(user_id);
88 }
```

Referenced by ReducerHandler::handle_reducer_job().

Here is the caller graph for this function:



Index

- ~AsyncClientCall
 - AsyncClientCall, [12](#)
- ~BaseHandler
 - BaseHandler, [15](#)
- ~Master
 - Master, [49](#)
- ~Worker
 - Worker, [78](#)
- ~WorkerClient
 - WorkerClient, [86](#)
- add_dependencies
 - CMakeLists.txt, [91](#)
- ALIVE
 - master.h, [104](#)
- anonymous_namespace{mr_task_factory.cc}, [9](#)
- anonymous_namespace{mr_task_factory.cc}::TaskFactory,
 - [74](#)
 - get_mapper, [75](#)
 - get_reducer, [75](#)
 - instance, [76](#)
 - mappers_, [76](#)
 - reducers_, [76](#)
 - TaskFactory, [75](#)
- approx_split
 - file_shard.h, [93](#)
- assign_files_to_reducer
 - Master, [49](#)
- assigned_partition
 - Master, [61](#)
- assigned_shards
 - Master, [61](#)
- async_map
 - Master, [50](#)
- async_reducer
 - Master, [51](#)
- AsyncClientCall, [11](#)
 - ~AsyncClientCall, [12](#)
 - context, [12](#)
 - is_map_job, [12](#)
 - status, [12](#)
 - worker_ip_addr, [13](#)
- BaseHandler, [13](#)
 - ~BaseHandler, [15](#)
 - BaseHandler, [14](#)
 - CallStatus, [14](#)
 - CREATE, [14](#)
 - ctx_, [16](#)
 - FINISH, [14](#)
 - Proceed, [15](#)
 - PROCESS, [14](#)
 - s_queue, [16](#)
 - service, [16](#)
 - status_, [16](#)
 - worker_address, [16](#)
- BaseMapperInternal, [17](#)
 - BaseMapperInternal, [17](#)
 - emit, [18](#)
 - final_flush, [19](#)
 - intermediate_file_list, [20](#)
 - internal_file_mapping, [19](#)
 - kv_pair_vector, [20](#)
- BaseReducerInternal, [21](#)
 - BaseReducerInternal, [21](#)
 - emit, [22](#)
 - file_name, [22](#)
- builder
 - Worker, [81](#)
- BUSY
 - master.h, [105](#)
- CallStatus
 - BaseHandler, [14](#)
- clean_exit
 - Worker, [81](#)
- cleanup_files
 - Master, [53](#)
- cleanup_mutex
 - Master, [61](#)
- client
 - worker, [83](#)
- cmake_minimum_required
 - CMakeLists.txt, [91](#)
- CMakeLists.txt
 - add_dependencies, [91](#)
 - cmake_minimum_required, [91](#)
- completion_count
 - Master, [62](#)
- condition_cleanup_mutex
 - Master, [62](#)
- condition_heartbeat
 - Master, [62](#)
- condition_ops_mutex
 - Master, [62](#)
- condition_worker_queue_mutex
 - Master, [62](#)
- context
 - AsyncClientCall, [12](#)
- convert_grpc_spec

- MapperHandler, 35
- WorkerClient, 86
- cq_
 - Master, 63
- CREATE
 - BaseHandler, 14
- create_shards
 - MapReduceImpl, 41
- ctx_
 - BaseHandler, 16
- current_output
 - worker, 83
- current_shard
 - worker, 83
- DEAD
 - master.h, 105
- dead_handled
 - worker, 83
- DEBUG
 - mr_tasks.h, 108
- DELIMITER
 - mr_tasks.h, 108
- devnull
 - mr_tasks.h, 108
- dummy
 - Master, 63
- emit
 - BaseMapperInternal, 18
 - BaseReducerInternal, 22
- file_name
 - BaseReducerInternal, 22
- file_shard.h
 - approx_split, 93
 - get_filesize, 94
 - KB, 93
 - shard_files, 94
 - TEMP_DIR, 93
- file_shards
 - Master, 63
- file_shards_
 - MapReduceImpl, 44
- filename
 - splitFile, 74
- FileShard, 23
 - shard_id, 23
 - split_file_list, 24
- final_flush
 - BaseMapperInternal, 19
- find_worker_by_name
 - Master, 54
- find_worker_by_status
 - Master, 55
- FINISH
 - BaseHandler, 14
- FREE
 - master.h, 105
- get_basemapper_internal
 - MapperHandler, 36
 - Worker, 78
- get_basereducer_internal
 - ReducerHandler, 69
 - Worker, 79
- get_filesize
 - file_shard.h, 94
- get_mapper
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 75
- get_mapper_from_task_factory
 - mr_task_factory.cc, 106
 - worker.h, 111
- get_reducer
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 75
- get_reducer_from_task_factory
 - mr_task_factory.cc, 106
 - worker.h, 111
- h_writer
 - HeartbeatHandler, 31
- handle_heartbeat_job
 - HeartbeatHandler, 29
- handle_mapper_job
 - MapperHandler, 37
- handle_reducer_job
 - ReducerHandler, 70
- handler_dead_worker
 - Master, 55
- heartbeat
 - Master, 57
- heartbeat_handler
 - Worker, 79
- heartbeat_mutex
 - Master, 63
- heartbeat_payload, 24
 - id, 25
 - timestamp, 25
 - workerStatus, 25
- heartbeat_payload_reader
 - HeartbeatCall, 26
- heartbeat_queue
 - Worker, 81
 - WorkerClient, 90
- HeartbeatCall, 26
 - heartbeat_payload_reader, 26
 - result, 27
- HeartbeatHandler, 27
 - h_writer, 31
 - handle_heartbeat_job, 29
 - HeartbeatHandler, 28
 - Proceed, 30
 - request, 31
 - response, 31
- id
 - heartbeat_payload, 25

- init_heartbeat
 - Master, 63
- input_files
 - MapReduceSpec, 45
- instance
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 76
- intermediate_file_list
 - BaseMapperInternal, 20
- intermidateFiles
 - Master, 64
- internal_file_mapping
 - BaseMapperInternal, 19
- is_map_job
 - AsyncClientCall, 12
- KB
 - file_shard.h, 93
- kv_pair_vector
 - BaseMapperInternal, 20
- m_writer
 - MapperHandler, 39
- main
 - run_worker.cc, 109
- map_kb
 - MapReduceSpec, 46
- map_response_reader
 - MapCall, 32
- MapCall, 32
 - map_response_reader, 32
 - result, 33
- MAPPER
 - master.h, 105
- MapperHandler, 33
 - convert_grpc_spec, 35
 - get_basemapper_internal, 36
 - handle_mapper_job, 37
 - m_writer, 39
 - MapperHandler, 34
 - mapRequest, 39
 - mapResponse, 39
 - Proceed, 38
- mappers_
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 76
- mapreduce_service
 - Worker, 81
- mapreduce_spec.h
 - read_mr_spec_from_config_file, 99
 - splitString, 101
 - validate_mr_spec, 102
- MapReduceImpl, 40
 - create_shards, 41
 - file_shards_, 44
 - mr_spec_, 44
 - read_and_validate_spec, 41
 - run, 42
 - run_master, 43
 - MapReduceSpec, 45
 - input_files, 45
 - map_kb, 46
 - output_directory, 46
 - output_files, 46
 - user, 46
 - worker_count, 46
 - worker_endpoints, 47
 - mapRequest
 - MapperHandler, 39
 - mapResponse
 - MapperHandler, 39
 - Master, 47
 - ~Master, 49
 - assign_files_to_reducer, 49
 - assigned_partition, 61
 - assigned_shards, 61
 - async_map, 50
 - async_reducer, 51
 - cleanup_files, 53
 - cleanup_mutex, 61
 - completion_count, 62
 - condition_cleanup_mutex, 62
 - condition_heartbeat, 62
 - condition_ops_mutex, 62
 - condition_worker_queue_mutex, 62
 - cq_, 63
 - dummy, 63
 - file_shards, 63
 - find_worker_by_name, 54
 - find_worker_by_status, 55
 - handler_dead_worker, 55
 - heartbeat, 57
 - heartbeat_mutex, 63
 - init_heartbeat, 63
 - intermidateFiles, 64
 - Master, 48
 - missing_output_files, 64
 - missing_shards, 64
 - mr_spec, 64
 - ops_completed, 64
 - ops_mutex, 65
 - OutputFiles, 65
 - run, 58
 - server_state, 65
 - worker_queue_mutex, 65
 - workers, 65
- master.h
 - ALIVE, 104
 - BUSY, 105
 - DEAD, 105
 - FREE, 105
 - MAPPER, 105
 - REDUCER, 105
 - TIMEOUT, 104
 - WORKER_STATUS, 104
 - WORKER_TYPE, 105
- MAX_KV_PAIR_SIZE

- mr_tasks.h, 109
- missing_output_files
 - Master, 64
- missing_shards
 - Master, 64
- mr_spec
 - Master, 64
- mr_spec_
 - MapReduceImpl, 44
- mr_task_factory.cc
 - get_mapper_from_task_factory, 106
 - get_reducer_from_task_factory, 106
 - register_tasks, 107
- mr_tasks.h
 - DEBUG, 108
 - DELIMITER, 108
 - devnull, 108
 - MAX_KV_PAIR_SIZE, 109
- offsets
 - splitFile, 74
- ops_completed
 - Master, 64
- ops_mutex
 - Master, 65
- output_directory
 - MapReduceSpec, 46
- output_files
 - MapReduceSpec, 46
- output_reducer_location_map
 - worker, 84
- OutputFiles
 - Master, 65
- Proceed
 - BaseHandler, 15
 - HeartbeatHandler, 30
 - MapperHandler, 38
 - ReducerHandler, 71
- PROCESS
 - BaseHandler, 14
- queue
 - WorkerClient, 90
- r_writer
 - ReducerHandler, 72
- read_and_validate_spec
 - MapReduceImpl, 41
- read_mr_spec_from_config_file
 - mapreduce_spec.h, 99
- recv_heartbeat
 - WorkerClient, 87
- ReduceCall, 66
 - reducer_response_reader, 67
 - result, 67
- REDUCER
 - master.h, 105
- reducer_response_reader
 - ReduceCall, 67
- ReduceRequest
 - ReducerHandler, 72
- ReduceResponse
 - ReducerHandler, 73
- ReducerHandler, 67
 - get_basereducer_internal, 69
 - handle_reducer_job, 70
 - Proceed, 71
 - r_writer, 72
 - ReduceRequest, 72
 - ReduceResponse, 73
 - ReducerHandler, 68
- reducers_
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 76
- register_tasks
 - mr_task_factory.cc, 107
- request
 - HeartbeatHandler, 31
- response
 - HeartbeatHandler, 31
- result
 - HeartbeatCall, 27
 - MapCall, 33
 - ReduceCall, 67
- run
 - MapReduceImpl, 42
 - Master, 58
 - Worker, 80
- run_master
 - MapReduceImpl, 43
- run_worker.cc
 - main, 109
- s_queue
 - BaseHandler, 16
- schedule_mapper_jobs
 - WorkerClient, 87
- schedule_reduce_job
 - WorkerClient, 88
- send_heartbeat
 - WorkerClient, 89
- server
 - Worker, 82
- server_state
 - Master, 65
- service
 - BaseHandler, 16
- shard_files
 - file_shard.h, 94
- shard_id
 - FileShard, 23
- split_file_list
 - FileShard, 24
- splitFile, 73
 - filename, 74
 - offsets, 74
- splitString

- mapreduce_spec.h, 101
- src/CMakeLists.txt, 91
- src/file_shard.h, 92
- src/mapreduce.cc, 96
- src/mapreduce_impl.cc, 97
- src/mapreduce_impl.h, 97
- src/mapreduce_spec.h, 98
- src/master.h, 103
- src/mr_task_factory.cc, 105
- src/mr_tasks.h, 107
- src/Readme.txt, 109
- src/run_worker.cc, 109
- src/worker.h, 110
- status
 - AsyncClientCall, 12
- status_
 - BaseHandler, 16
- stub
 - WorkerClient, 90
- TaskFactory
 - anonymous_namespace{mr_task_factory.cc}::TaskFactory, 75
- TEMP_DIR
 - file_shard.h, 93
- TIMEOUT
 - master.h, 104
- timestamp
 - heartbeat_payload, 25
- user
 - MapReduceSpec, 46
- validate_mr_spec
 - mapreduce_spec.h, 102
- work_queue
 - Worker, 82
- Worker, 77
 - ~Worker, 78
 - builder, 81
 - clean_exit, 81
 - get_basemapper_internal, 78
 - get_basereducer_internal, 79
 - heartbeat_handler, 79
 - heartbeat_queue, 81
 - mapreduce_service, 81
 - run, 80
 - server, 82
 - work_queue, 82
 - Worker, 78
 - worker_uuid, 82
- worker, 82
 - client, 83
 - current_output, 83
 - current_shard, 83
 - dead_handled, 83
 - output_reducer_location_map, 84
 - worker_address, 84
 - workerStatus, 84
 - workerType, 84
- worker.h
 - get_mapper_from_task_factory, 111
 - get_reducer_from_task_factory, 111
- worker_address
 - BaseHandler, 16
 - worker, 84
 - WorkerClient, 90
- worker_count
 - MapReduceSpec, 46
- worker_endpoints
 - MapReduceSpec, 47
- worker_ip_addr
 - AsyncClientCall, 13
- worker_queue_mutex
 - Master, 65
- WORKER_STATUS
 - master.h, 104
- WORKER_TYPE
 - master.h, 105
- worker_uuid
 - Worker, 82
- WorkerClient, 85
 - ~WorkerClient, 86
 - convert_grpc_spec, 86
 - heartbeat_queue, 90
 - queue, 90
 - recv_heartbeat, 87
 - schedule_mapper_jobs, 87
 - schedule_reduce_job, 88
 - send_heartbeat, 89
 - stub, 90
 - worker_address, 90
 - WorkerClient, 86
- workers
 - Master, 65
- workerStatus
 - heartbeat_payload, 25
 - worker, 84
- workerType
 - worker, 84