Project 4 MapReduce

1

Generated by Doxygen 1.8.17

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 anonymous_namespace{mr_task_factory.cc} Namespace Reference

Data Structures

- class TaskFactory

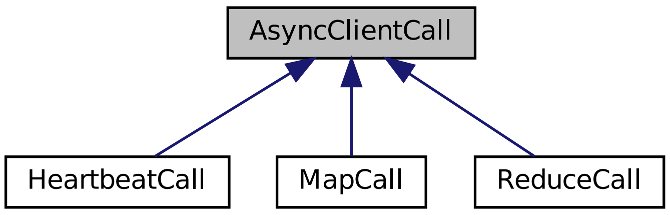
Chapter 6

Data Structure Documentation

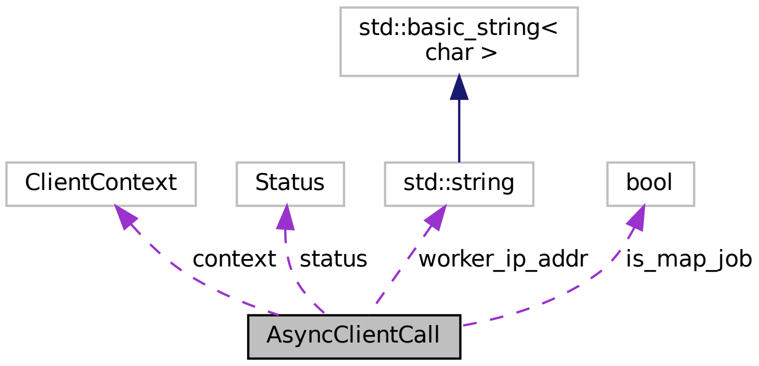
6.1 AsyncClientCall Class Reference

```
#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().

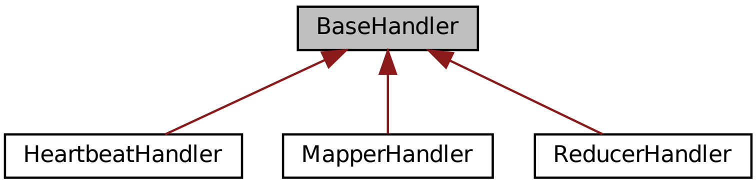
The documentation for this class was generated from the following file:

- src/master.h

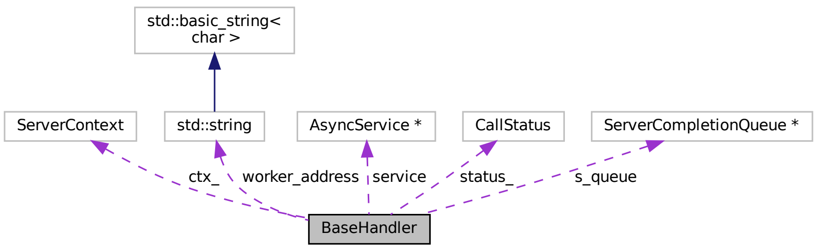
6.2 BaseHandler Class Reference

```
#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.

```
      {
          CREATE,
          PROCESS,
          FINISH
      };
```

6.2.3 Constructor & Destructor Documentation

6.2.3.1 BaseHandler()

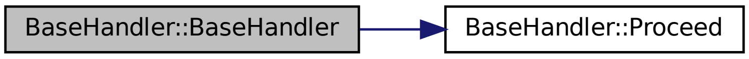
```
BaseHandler::BaseHandler (
            masterworker::Map_Reduce::AsyncService * service,
            grpc::ServerCompletionQueue * queue,
            std::string worker_address )  [inline]
```

Definition at line 28 of file worker.h.

```
        : service(service)
        , s_queue(queue)
        , worker_address(std::move(worker_address))
        , status_(CREATE)
    {
        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]
```

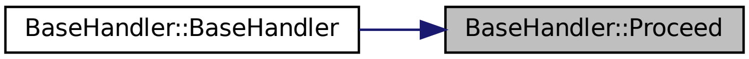
Reimplemented in HeartbeatHandler, ReducerHandler, and MapperHandler.

Definition at line 40 of file worker.h.

```
    {
    }
```

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().

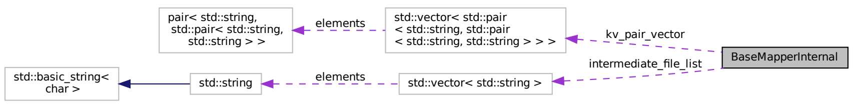
The documentation for this class was generated from the following file:

- src/worker.h

6.3 BaseMapperInternal Struct Reference

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

Collaboration diagram for BaseMapperInternal:


pair< std::string, std::pair< std::string, std::string > >
elements
std::vector< std::pair < std::string, std::pair < std::string, std::string > > >
kv_pair_vector
BaseMapperInternal
intermediate_file_list
std::basic_string< char >
std::string
elements
std::vector< std::string >

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

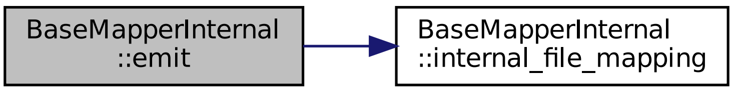
key	
val	

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
      « DELIMITER
71 //    « val « std::endl;
72 #endif
73     if (BaseMapperInternal::kv_pair_vector.size() > MAX_KV_PAIR_SIZE)
74     {
75         for (const auto& a : BaseMapperInternal::kv_pair_vector)
76         {
77             std::ofstream f(a.first, std::ofstream::out | std::ofstream::app);
78             f « a.second.first « DELIMITER « a.second.second « std::endl;
79         }
80         BaseMapperInternal::kv_pair_vector.clear();
81     }
82     BaseMapperInternal::kv_pair_vector.push_back({BaseMapperInternal::internal_file_mapping(key), {key,
      val}});
83 }
```

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

Here is the call graph for this function:


BaseMapperInternal ::emit
BaseMapperInternal ::internal_file_mapping

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.

```
{
    for (const auto& a : BaseMapperInternal::kv_pair_vector)
    {
        std::ofstream f(a.first, std::ofstream::out | std::ofstream::app);
        f « a.second.first « DELIMITER « a.second.second « std::endl;
        f.close();
    }
    BaseMapperInternal::kv_pair_vector.clear();
}
```

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.

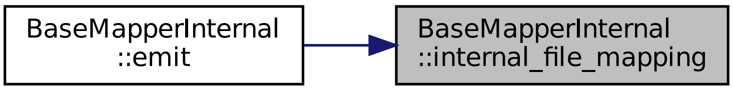
Definition at line 53 of file mr_tasks.h.

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

References devnull, and intermediate_file_list.

Referenced by emit().

Here is the caller graph for this function:


BaseMapperInternal ::emit
BaseMapperInternal ::internal_file_mapping

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> > > BaseMapper↩
Internal::kv_pair_vector
```

Storage vector for key value pairs

Definition at line 30 of file mr_tasks.h.

Referenced by emit(), and final_flush().

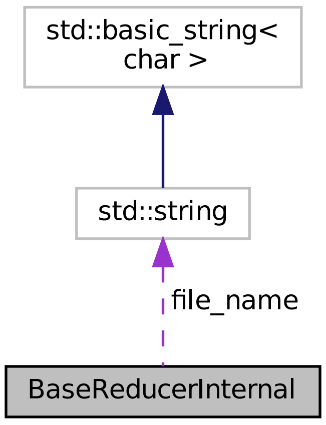
The documentation for this struct was generated from the following file:

- 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

key	
val	

Definition at line 129 of file mr_tasks.h.

```
{
#if DEBUG > 1
    std::cout « "Dummy emit by BaseReducerInternal: " « key « ", " « val « std::endl;
#endif
    std::ofstream f(file_name, std::ofstream::out | std::ofstream::app);
    f « key « " " « val « std::endl;
    f.close();
}
```

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().

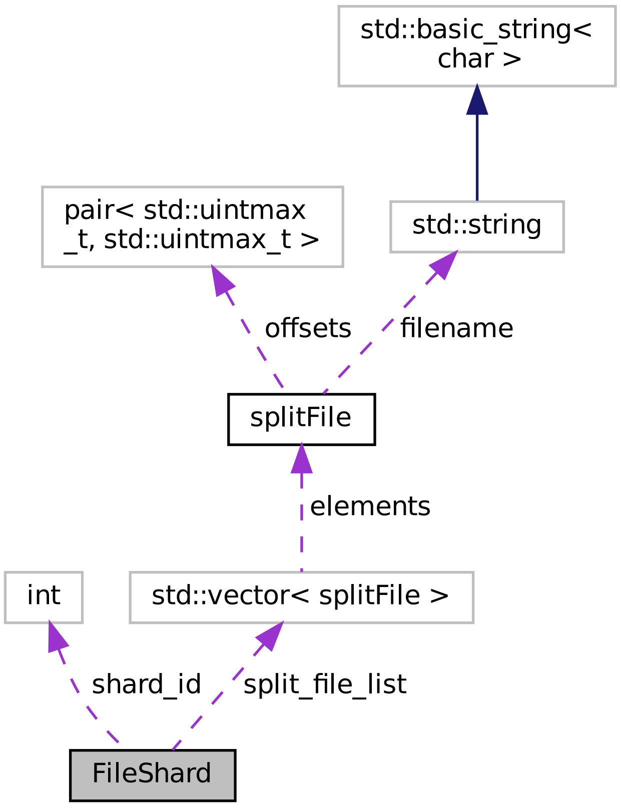
The documentation for this struct was generated from the following file:

- 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().

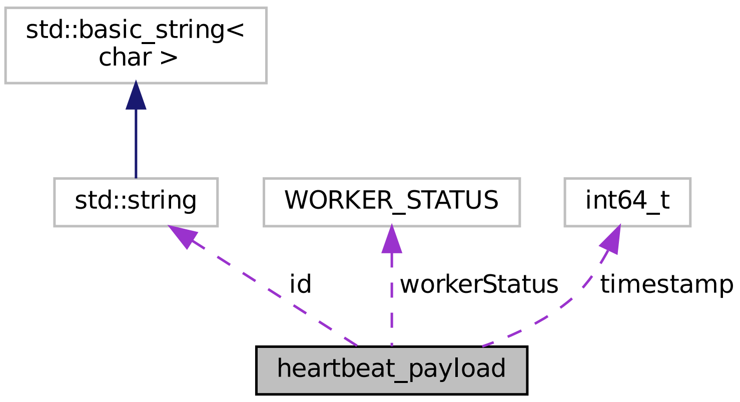
The documentation for this struct was generated from the following file:

- 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.

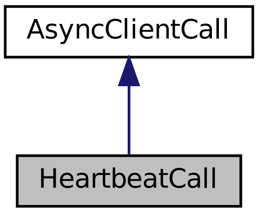
The documentation for this struct was generated from the following file:

- src/master.h

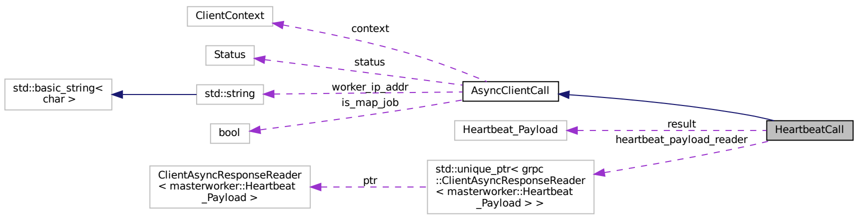
6.7 HeartbeatCall Class Reference

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

Inheritance diagram for HeartbeatCall:



Collaboration diagram for HeartbeatCall:


ClientContext
context
Status
status
std::basic_string< char >
std::string
worker_ip_addr
is_map_job
AsyncClientCall
bool
Heartbeat_Payload
result
heartbeat_payload_reader
HeartbeatCall
ClientAsyncResponseReader < masterworker::Heartbeat _Payload >
ptr
std::unique_ptr< grpc ::ClientAsyncResponseReader < masterworker::Heartbeat _Payload > >

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> > Heartbeat↩
Call::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.

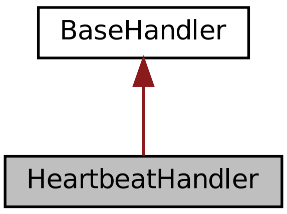
The documentation for this class was generated from the following file:

- src/master.h

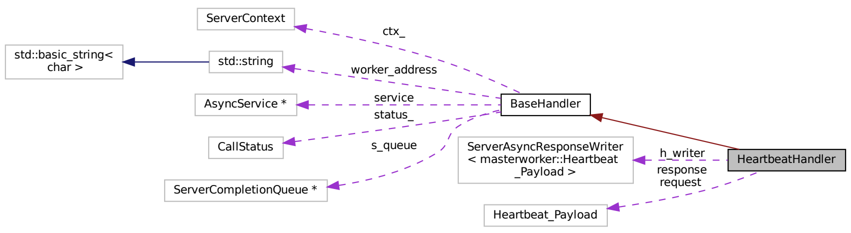
6.8 HeartbeatHandler Class Reference

```
#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

service	
pQueue	
basicString	

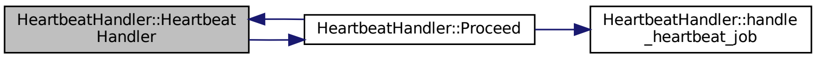
Definition at line 177 of file worker.h.

```
        : BaseHandler(service, pQueue, basicString)
        , h_writer(&ctx_)
    {
        HeartbeatHandler::Proceed();
    }
```

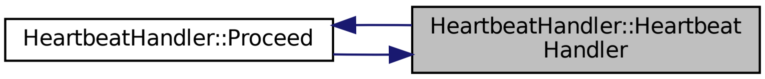
References Proceed().

Referenced by Proceed().

Here is the call graph for this function:


HeartbeatHandler::Heartbeat Handler
HeartbeatHandler::Proceed
HeartbeatHandler::handle _heartbeat_job

Here is the caller graph for this function:


HeartbeatHandler::Proceed
HeartbeatHandler::Heartbeat Handler

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

request	Heartbeat payload check .proto

Returns

Heartbeat payload

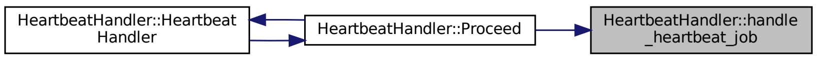
Definition at line 432 of file worker.h.

```
{
    masterworker::Heartbeat_Payload payload;
    payload.set_id(request.id());
    if (true)
        payload.set_status(masterworker::Heartbeat_Payload_type_ALIVE);
    return payload;
}
```

References request.

Referenced by Proceed().

Here is the caller graph for this function:


HeartbeatHandler::Heartbeat Handler
HeartbeatHandler::Proceed
HeartbeatHandler::handle _heartbeat_job

6.8.3.2 Proceed()

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

Reimplemented from BaseHandler.

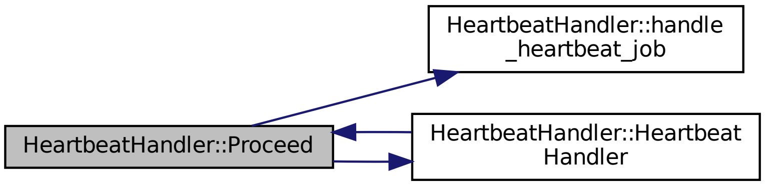
Definition at line 187 of file worker.h.

```
    {
        if (status_ == CREATE)
        {
            status_ = PROCESS;
            service->Requestheartbeat(&ctx_, &request, &h_writer, s_queue, s_queue, this);
        }
        else if (status_ == PROCESS)
        {
            new HeartbeatHandler(service, s_queue, worker_address);
            response = handle_heartbeat_job(request);
            status_ = FINISH;
            h_writer.Finish(response, grpc::Status::OK, this);
        }
        else
        {
            GPR_ASSERT(status_ == FINISH);
            delete this;
        }
    }
```

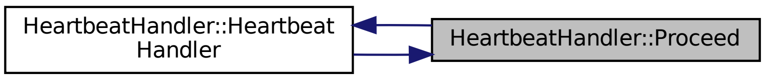
References BaseHandler::CREATE, BaseHandler::ctx_, BaseHandler::FINISH, h_writer, handle_heartbeat_job(), HeartbeatHandler(), BaseHandler::PROCESS, request, response, BaseHandler::s_queue, BaseHandler::service, BaseHandler::status_, and BaseHandler::worker_address.

Referenced by HeartbeatHandler().

Here is the call graph for this function:


HeartbeatHandler::handle _heartbeat_job
HeartbeatHandler::Proceed
HeartbeatHandler::Heartbeat Handler

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().

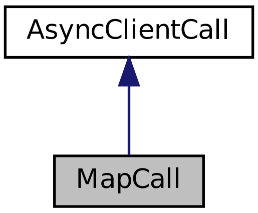
The documentation for this class was generated from the following file:

- src/worker.h

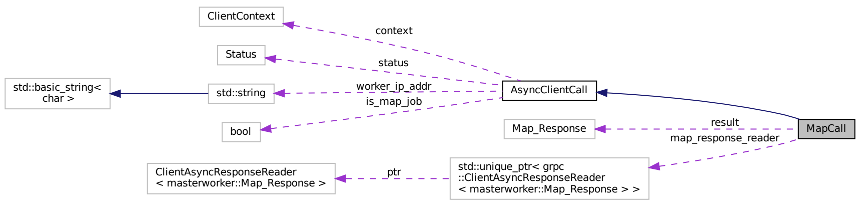
6.9 MapCall Class Reference

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

Inheritance diagram for MapCall:



Collaboration diagram for MapCall:


ClientContext
context
Status
status
std::basic_string< char >
std::string
worker_ip_addr
is_map_job
AsyncClientCall
bool
Map_Response
result
map_response_reader
MapCall
ClientAsyncResponseReader < masterworker::Map_Response >
ptr
std::unique_ptr< grpc ::ClientAsyncResponseReader < masterworker::Map_Response > >

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.

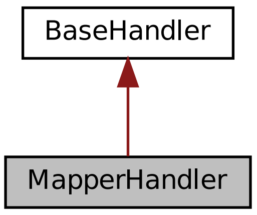
The documentation for this class was generated from the following file:

- src/master.h

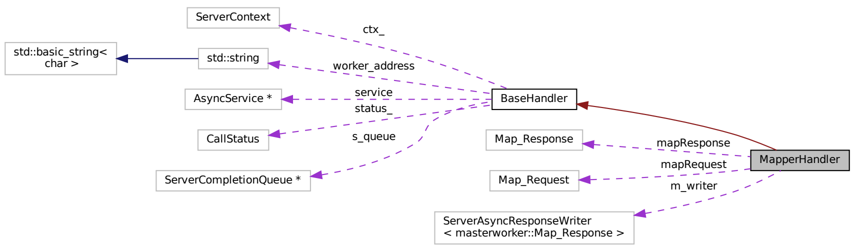
6.10 MapperHandler Class Reference

```
#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

service	
pQueue	
basicString	

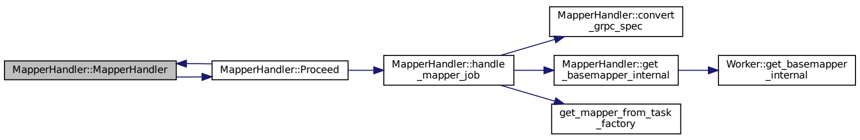
Definition at line 72 of file worker.h.

```
        : BaseHandler(service, pQueue, basicString)
        , m_writer(&ctx_)
    {
        Proceed();
    }
```

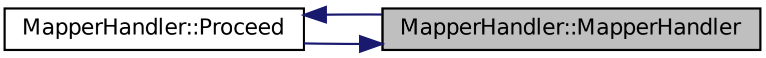
References Proceed().

Referenced by Proceed().

Here is the call graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle_mapper_job
MapperHandler::convert_grpc_spec
MapperHandler::get_basemapper_internal
Worker::get_basemapper_internal
get_mapper_from_task_factory

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

partition	

Returns

FileShard struct equivalent of gprc partition payload.

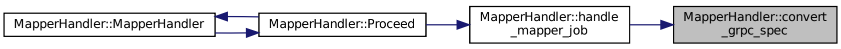
Definition at line 325 of file worker.h.

```
326 {
    FileShard shard{};
    shard.shard_id = partition.shard_id();
    for (auto f : partition.file_list())
    {
        splitFile temp{};
        temp.filename = f.filename();
        temp.offsets = {f.start_offset(), f.end_offset()};
        shard.split_file_list.push_back(temp);
    }
    return shard;
}
```

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

Referenced by handle_mapper_job().

Here is the caller graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle_mapper_job
MapperHandler::convert_grpc_spec

6.10.3.2 get_basemapper_internal()

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

Parameters

mapper	

Returns

BaseMapperinternal Class

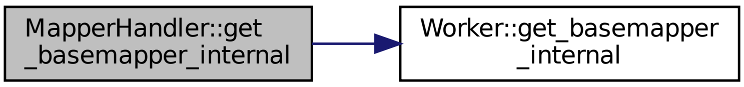
Definition at line 454 of file worker.h.

```
{
    return Worker::get_basemapper_internal(mapper);
}
```

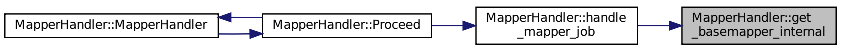
References Worker::get_basemapper_internal().

Referenced by handle_mapper_job().

Here is the call graph for this function:


MapperHandler::get_basemapper_internal
Worker::get_basemapper_internal

Here is the caller graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle_mapper_job
MapperHandler::get_basemapper_internal

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 intermedidate file. usually TEMP_DIR/<partition_count>_<worker_port>.txt Reads shard files and apply user defined map function on it.

Parameters

request	grpc Map Request , check .proto

Returns

grpc Map Response payload , check .proto

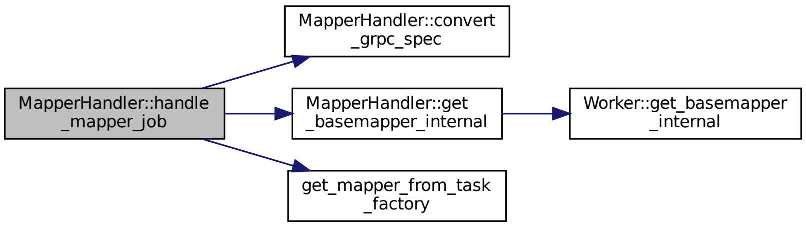
Definition at line 344 of file worker.h.

```
{
    masterworker::Map_Response payload;
    auto user_mapper_func = get_mapper_from_task_factory(request.uuid());
    auto base_mapper = get_basemapper_internal(user_mapper_func.get());
    auto partition_count = request.partition_count();
    base_mapper->intermediate_file_list.reserve(partition_count);
    for (int i = 0; i < partition_count; i++)
    {
        base_mapper->intermediate_file_list.push_back(std::string(
            std::string(TEMP_DIR) + "/" + std::to_string(i) + "_" + MapperHandler::worker_address +
   ".txt"));
    }
    FileShard local_shard;
    for (int shard_count = 0; shard_count < request.shard_size(); shard_count++)
    {
        local_shard = MapperHandler::convert_grpc_spec(request.shard(shard_count));

        for (const auto& i : local_shard.split_file_list)
        {
            std::string mapper_line;
            std::ifstream f(i.filename, std::ios::binary);
            if (!f.good())
            {
                std::cerr « i.filename « " not open...." « std::endl;
            }

            f.seekg(i.offsets.first);
            std::string dummy(i.offsets.second - i.offsets.first, ' ');
            f.read(&dummy[0], i.offsets.second - i.offsets.first);
            std::stringstream stream(dummy);
            while (std::getline(stream, mapper_line))
            {
                user_mapper_func->map(mapper_line);
            }
        }
    }
    base_mapper->final_flush();
    for (const auto& i : base_mapper->intermediate_file_list)
    {
        payload.add_file_list(i);
    }

    return payload;
}
```

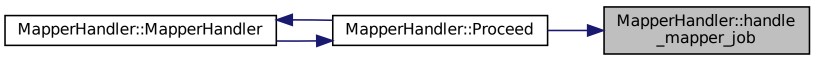
References convert_grpc_spec(), get_basemapper_internal(), get_mapper_from_task_factory(), FileShard::split↩_file_list, TEMP_DIR, and BaseHandler::worker_address.

Referenced by Proceed().

Here is the call graph for this function:


MapperHandler::convert _grpc_spec
MapperHandler::handle _mapper_job
MapperHandler::get _basemapper_internal
Worker::get_basemapper _internal
get_mapper_from_task _factory

Here is the caller graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle _mapper_job

6.10.3.4 Proceed()

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

Reimplemented from BaseHandler.

Definition at line 82 of file worker.h.

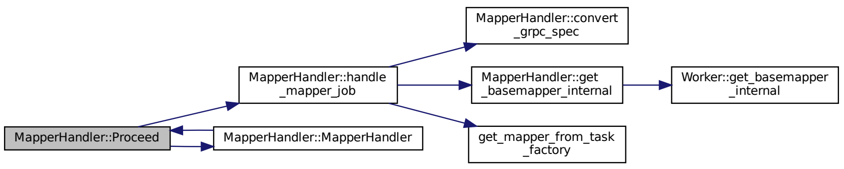
```
    {
        if (status_ == CREATE)
        {
            status_ = PROCESS;
            service->Requestmap(&ctx_, &mapRequest, &m_writer, s_queue, s_queue, this);
        }
        else if (status_ == PROCESS)
        {
            new MapperHandler(service, s_queue, worker_address);
            mapResponse = handle_mapper_job(mapRequest);
            status_ = FINISH;
            m_writer.Finish(mapResponse, grpc::Status::OK, this);
        }
        else
        {
            GPR_ASSERT(status_ == FINISH);
            delete this;
         }
```

```
101     }
```

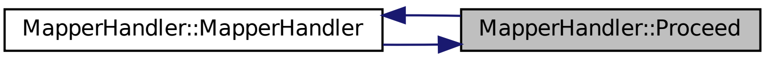
References BaseHandler::CREATE, BaseHandler::ctx_, BaseHandler::FINISH, handle_mapper_job(), m_writer, MapperHandler(), mapRequest, mapResponse, BaseHandler::PROCESS, BaseHandler::s_queue, BaseHandler↩ ::service, BaseHandler::status_, and BaseHandler::worker_address.

Referenced by MapperHandler().

Here is the call graph for this function:


MapperHandler::convert _grpc_spec
MapperHandler::handle _mapper_job
MapperHandler::get _basemapper_internal
Worker::get_basemapper _internal
MapperHandler::Proceed
MapperHandler::MapperHandler
get_mapper_from_task _factory

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().

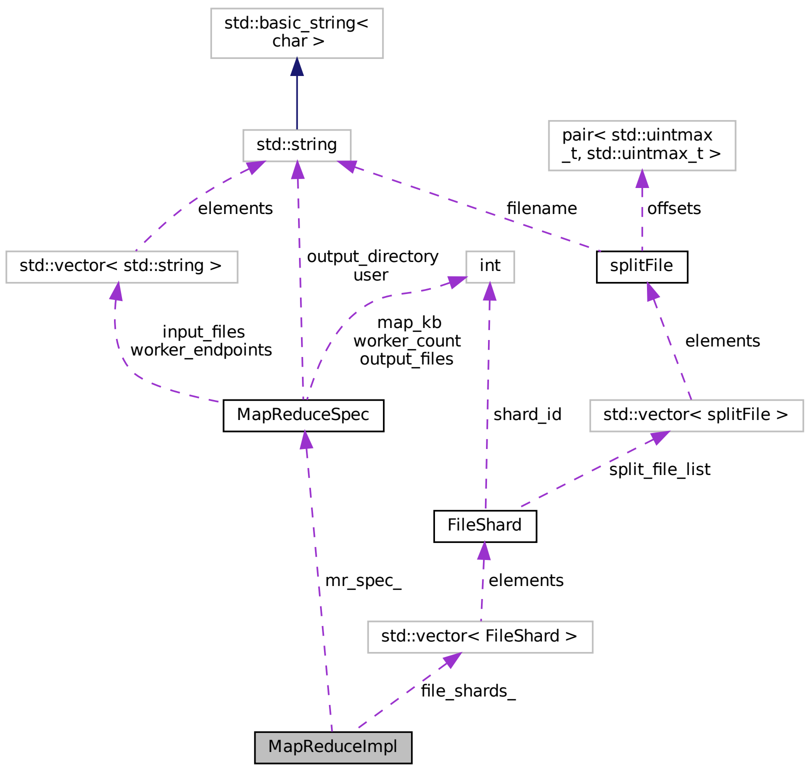
The documentation for this class was generated from the following file:

- 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]
```

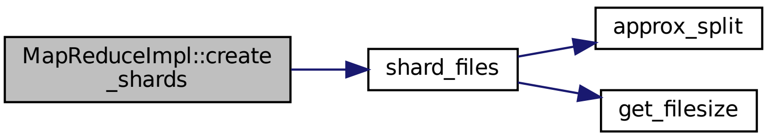
Definition at line 39 of file mapreduce_impl.cc.

```
{
    return shard_files(mr_spec_, file_shards_);
}
```

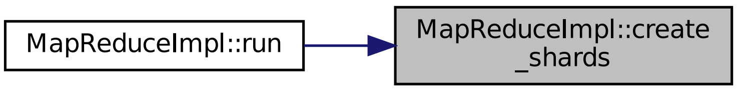
References file_shards_, mr_spec_, and shard_files().

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]
```

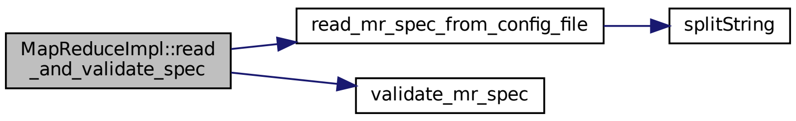
Definition at line 33 of file mapreduce_impl.cc.

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

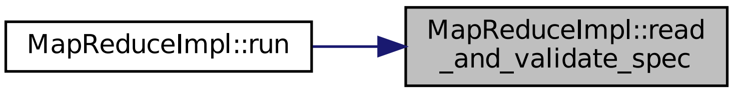
References mr_spec_, read_mr_spec_from_config_file(), and validate_mr_spec().

Referenced by run().

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.2.3 run()

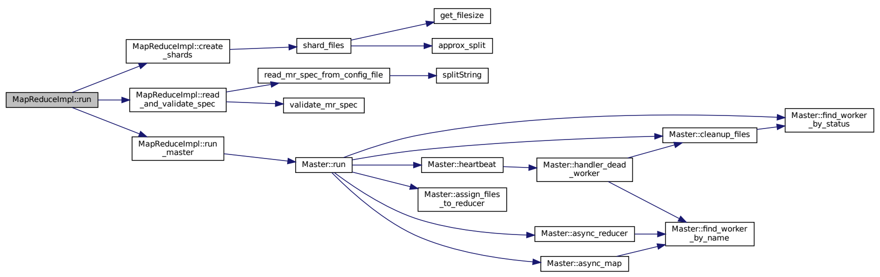
```
bool MapReduceImpl::run (
            const std::string & config_filename )
```

Definition at line 8 of file mapreduce_impl.cc.

```
{

    if (!read_and_validate_spec(config_filename))
    {
        std::cerr « "Spec not configured properly." « std::endl;
        return false;
    }

    if (!create_shards())
    {
        std::cerr « "Failed to create shards." « std::endl;
        return false;
    }

    if (!run_master())
    {
        std::cerr « "MapReduce failure. Something didn’t go well!" « std::endl;
        return false;
    }

    return true;
}
```

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

Here is the call graph for this function:


MapReduceImpl::run
MapReduceImpl::create_shards
MapReduceImpl::read_and_validate_spec
MapReduceImpl::run_master
shard_files
get_filesize
approx_split
read_mr_spec_from_config_file
splitString
validate_mr_spec
Master::run
Master::find_worker_by_status
Master::cleanup_files
Master::heartbeat
Master::handler_dead_worker
Master::assign_files_to_reducer
Master::async_reducer
Master::find_worker_by_name
Master::async_map

6.11.2.4 run_master()

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

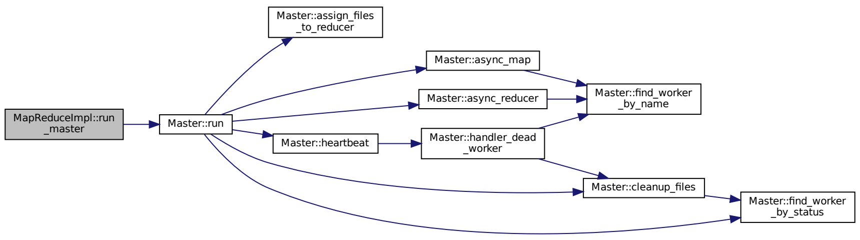
Definition at line 45 of file mapreduce_impl.cc.

```
{
    Master master(mr_spec_, file_shards_);
    return master.run();
}
```

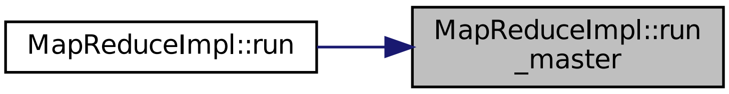
References file_shards_, mr_spec_, and Master::run().

Referenced by run().

Here is the call graph for this function:


MapReduceImpl::run_master
Master::run
Master::assign_files_to_reducer
Master::async_map
Master::async_reducer
Master::heartbeat
Master::handler_dead_worker
Master::find_worker_by_name
Master::cleanup_files
Master::find_worker_by_status

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().

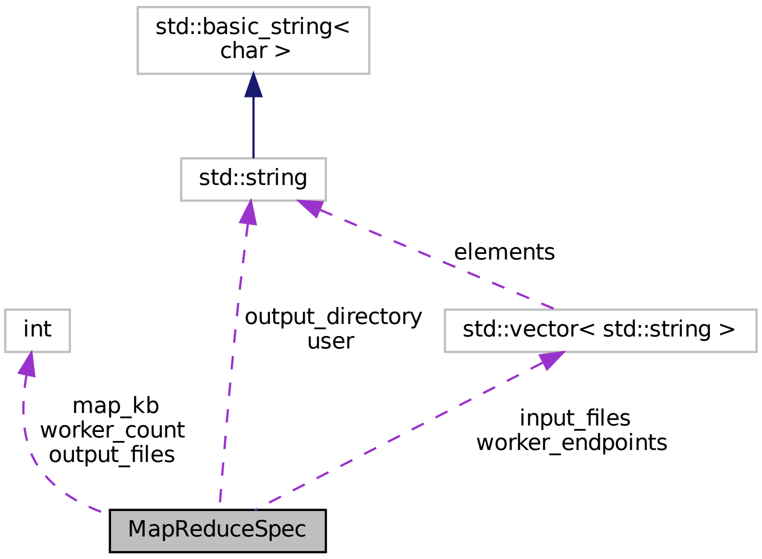
The documentation for this class was generated from the following files:

- 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().

6.12.2.6 worker_count

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

Definition at line 25 of file mapreduce_spec.h.

Referenced by read_mr_spec_from_config_file(), and validate_mr_spec().

6.12.2.7 worker_endpoints

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

Definition at line 30 of file mapreduce_spec.h.

Referenced by read_mr_spec_from_config_file(), and validate_mr_spec().

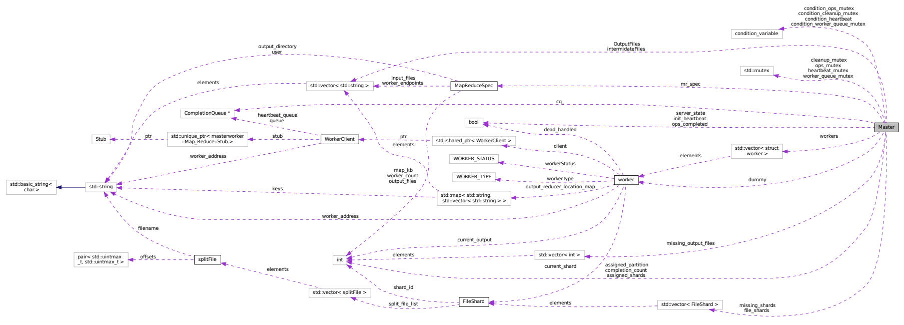
The documentation for this struct was generated from the following file:

- src/mapreduce_spec.h

6.13 Master Class Reference

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

Collaboration diagram for Master:



Public Member Functions

- 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

mr_spec	
file_shards	

Definition at line 363 of file master.h.

```
    : mr_spec(mr_spec)
    , file_shards(file_shards)
{
    cq_ = new grpc::CompletionQueue();
    for (const auto& i : Master::mr_spec.worker_endpoints)
    {
        dummy.worker_address = i;
        dummy.workerStatus = FREE;
        dummy.workerType = MAPPER;
        dummy.client = std::make_shared<WorkerClient>(i, Master::cq_);
        Master::workers.push_back(dummy);
    }
}
```

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

6.13.2.2 ∼Master()

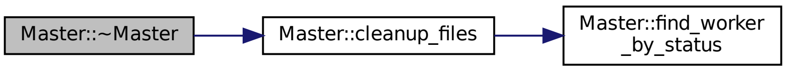
```
Master::∼Master ( )  [inline]
```

Definition at line 262 of file master.h.

```
    {
        Master::server_state = !ALIVE;
        Master::cq_->Shutdown();
        cleanup_files();
    }
```

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 intermidiate files to output files for reducing

Parameters

output←_id	

Returns

list of intermediate files for given output id /file

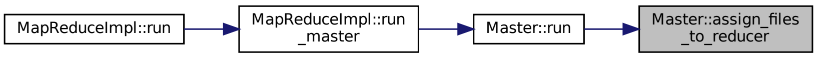
Definition at line 815 of file master.h.

```
{
    std::set<std::string> file_list;
    for (int i = 0; i < Master::intermidateFiles.size(); i++)
    {
        auto f = Master::intermidateFiles[i];
        if (i % Master::mr_spec.output_files == output_id)
        {
            file_list.insert(f);
        }
    }
    std::vector<std::string> convert;
    convert.assign(file_list.begin(), file_list.end());
    return convert;
}
```

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

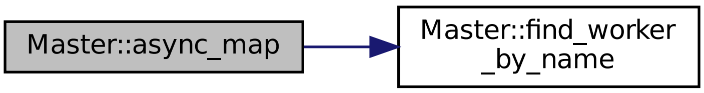
Definition at line 706 of file master.h.

```
{
    void* tag;
    bool ok = false;
    while (Master::cq_->Next(&tag, &ok))
    {
        auto call = static_cast<AsyncClientCall*>(tag);
        if (call->status.ok())
        {
            if (Master::find_worker_by_name(call->worker_ip_addr)->workerStatus != DEAD)
            {
                {
                    std::lock_guard<std::mutex> worker_queue(this->worker_queue_mutex);
                    for (auto& worker : Master::workers)
                    {
                        if (worker.worker_address == call->worker_ip_addr)
                        {
                            std::cout « call->worker_ip_addr + " back to free." « std::endl;

                            worker.workerStatus = FREE;
                            Master::completion_count--;
                            std::cout « call->worker_ip_addr + " response recieved. Completion Count : "
    +
                                             std::to_string(Master::completion_count) +
                                             " Assigned Work: " +
    std::to_string(Master::assigned_shards)
                                      « std::endl;
                            break;
                        }
                    }
                    condition_worker_queue_mutex.notify_one();
                }
                if (call->is_map_job)
                {
                    auto mcall = dynamic_cast<MapCall*>(call);
                    for (const auto& m : mcall->result.file_list())
                    {
                        Master::intermidateFiles.push_back(m);
                    }
                }
                {
                    std::unique_lock<std::mutex> work_done(ops_mutex);
                    if (Master::completion_count == 0)
                    {
                        ops_completed = true;
                        condition_ops_mutex.notify_one();
                        break;
                    }
                }
            }
            condition_cleanup_mutex.notify_one();
        }
        delete call;
    }
}
```

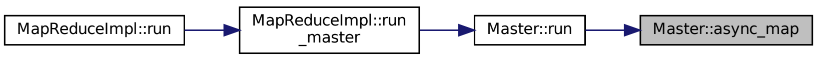
References assigned_shards, completion_count, condition_cleanup_mutex, condition_ops_mutex, condition_↩ worker_queue_mutex, cq_, DEAD, find_worker_by_name(), FREE, intermidateFiles, ops_completed, ops_mutex, worker::worker_address, worker_queue_mutex, workers, and worker::workerStatus.

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]
```

Simillar Async_map , handles Reducer Responses.

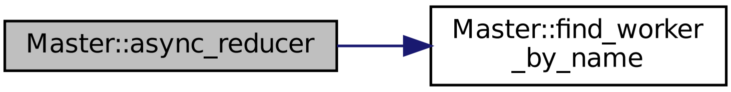
Definition at line 762 of file master.h.

```
{
    void* tag;
    bool ok = false;
    while (Master::cq_->Next(&tag, &ok))
    {
        auto call = static_cast<AsyncClientCall*>(tag);
        if (call->status.ok())
        {
            if (Master::find_worker_by_name(call->worker_ip_addr)->workerStatus != DEAD)
            {
                {
                    std::lock_guard<std::mutex> worker_queue(this->worker_queue_mutex);
                    for (auto& worker : Master::workers)
                    {
                        if (worker.worker_address == call->worker_ip_addr)
                        {
                            worker.workerStatus = FREE;
                            Master::completion_count--;
                            std::cout « call->worker_ip_addr + " response received. Completion Count : "
       +
                                                 std::to_string(Master::completion_count) +
                                                 " Assigned Work: " +
       std::to_string(Master::assigned_partition)
                                          « std::endl;
                            break;
                        }
                    }
                    condition_worker_queue_mutex.notify_one();
                }
                if (!call->is_map_job)
                {
                    auto mcall = dynamic_cast<ReduceCall*>(call);
                    Master::OutputFiles.push_back(mcall->result.file_name());
                }
                {
                    std::unique_lock<std::mutex> work_done(ops_mutex);
                    if (Master::completion_count == 0)
                    {
                        ops_completed = true;
                        condition_ops_mutex.notify_one();
                        break;
                    }
                }
            }
            condition_cleanup_mutex.notify_one();
        }
        delete call;
    }
}
```

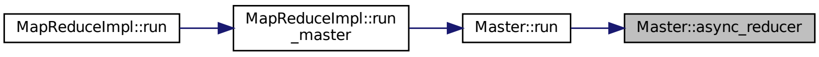
References assigned_partition, completion_count, condition_cleanup_mutex, condition_ops_mutex, condition↩_worker_queue_mutex, cq_, DEAD, find_worker_by_name(), FREE, ops_completed, ops_mutex, OutputFiles, worker::worker_address, worker_queue_mutex, workers, and worker::workerStatus.

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.

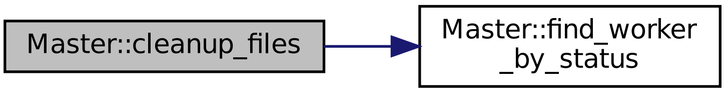
Definition at line 653 of file master.h.

```
{
    if (!Master::find_worker_by_status(DEAD).empty() && Master::server_state == ALIVE)
    {
        for (auto i : Master::find_worker_by_status(DEAD))
        {
            if (Master::workers[i].workerType == MAPPER && !Master::workers[i].dead_handled)
            {
                // std::string(TEMP_DIR) + "/" + std::to_string(i) + "_" + MapperHandler::worker_address
       + ".txt"))

                auto worker_port =

       Master::workers[i].worker_address.substr(Master::workers[i].worker_address.find_first_of(':'));
#if __cplusplus >= 201703L
                for (auto f : fs::directory_iterator(TEMP_DIR))
                {
                    if (f.path().string().find(worker_port) != std::string::npos)
                        fs::remove(f);
                }
#else
                auto dir_string = std::string("rm -rf ") + TEMP_DIR + "/*_" + worker_port + ".txt";
                system(dir_string.c_str());
#endif

            }
            else
            {
                for (const auto& worker_location : Master::workers[i].output_reducer_location_map)
                {
                    if ( !Master::workers[i].dead_handled)
#if __cplusplus >= 201703L

                        fs::remove(worker_location.first);
#else

                        remove(worker_location.first.c_str());
#endif
                }
            }
            Master::workers[i].dead_handled = true;
        }
    }
    if (!Master::server_state)
    {
#if __cplusplus >= 201703L
        fs::remove_all(TEMP_DIR);
#else
        auto dir_string = std::string("rm -rf " )+ TEMP_DIR;
        system(dir_string.c_str());
#endif
    }
}
```

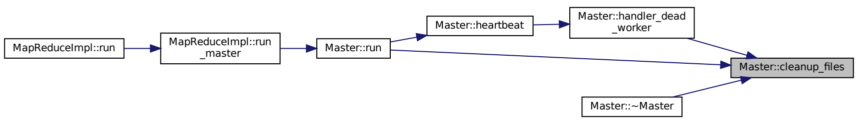
References ALIVE, DEAD, find_worker_by_status(), MAPPER, server_state, TEMP_DIR, and workers.

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

t	

Returns

pointer for Worker with requested name or Null pointer if not found

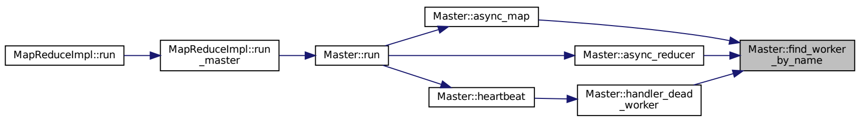
Definition at line 343 of file master.h.

```
{
    for (auto& w : Master::workers)
    {
        if (w.worker_address == t)
            return &w;
    }
    std::cerr « "worker " « t « " not found" « std::endl;
    return nullptr;
}
```

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

t	

Returns

list of Worker id with requested status

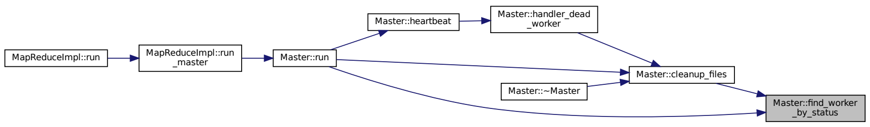
Definition at line 325 of file master.h.

```
{
    std::vector<int> temp;
    for (int i = 0; i < Master::workers.size(); i++)
    {
        if (Master::workers[i].workerStatus == t)
        {
            temp.push_back(i);
        }
    }
    return temp;
}
```

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

worker	address/id

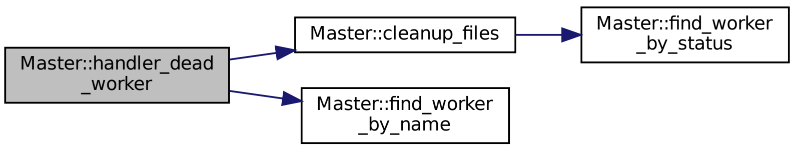
Definition at line 611 of file master.h.

```
{
    std::cerr « "HANDLING DEAD WORKER......" + worker « std::endl;
    auto w = Master::find_worker_by_name(worker);
    // Handle Mapper
    if (w->workerType == MAPPER and !ops_completed)
    {
        std::lock_guard<std::mutex> lockGuard(Master::cleanup_mutex);
        auto c = w->client.get();
        //        unintialized....
        if (!c)
        {
            condition_ops_mutex.notify_all();
            return;
        }
        Master::missing_shards.push_back(w->current_shard);
        Master::assigned_shards++;
        w->workerStatus = DEAD;
        Master::cleanup_files();
        condition_cleanup_mutex.notify_one();
    }
    else
    {
        std::lock_guard<std::mutex> lockGuard(Master::cleanup_mutex);
        auto c = w->client.get();
        if (!c)
        {
            condition_ops_mutex.notify_all();
            return;
        }
        Master::missing_output_files.push_back(w->current_output);
        Master::assigned_partition++;
        w->workerStatus = DEAD;
        Master::cleanup_files();
        condition_cleanup_mutex.notify_one();
    }
    condition_ops_mutex.notify_all();
}
```

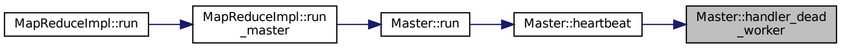
References assigned_partition, assigned_shards, cleanup_files(), cleanup_mutex, condition_cleanup_mutex, condition_ops_mutex, DEAD, find_worker_by_name(), MAPPER, missing_output_files, missing_shards, and ops↩_completed.

Referenced by heartbeat().

Here is the call graph for this function:


Master::handler_dead_worker
Master::cleanup_files
Master::find_worker_by_status
Master::find_worker_by_name

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.

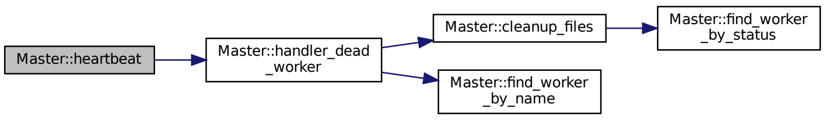
Definition at line 550 of file master.h.

```
{
    while (Master::server_state)
    {
        std::map<std::string, heartbeat_payload> message_queue;
        auto current_time = std::chrono::system_clock::now().time_since_epoch().count();
        for (const auto& w : Master::workers)
        {
            auto c = w.client.get();
            heartbeat_payload temp_payload{};
            temp_payload.id = w.worker_address;
            //            temp_payload.timestamp = current_time;
            if (w.workerStatus != DEAD)
            {
                c->send_heartbeat(temp_payload.timestamp);
                message_queue[w.worker_address] = temp_payload;
            }
        }

        for (auto& w : this->workers)
        {
            auto c = w.client.get();
            if (!c)
                continue;
            if (w.workerStatus != DEAD)
            {
                bool status = c->recv_heartbeat();
                if (!status)
                {
                    std::cerr « "Error " « w.worker_address « " : Dead , cleaning up" « std::endl;
                    w.workerStatus = DEAD;
                    Master::handler_dead_worker(message_queue[w.worker_address].id);
                }
            }
        }

        if (init_heartbeat)
        {
            {
                std::unique_lock<std::mutex> heartbeat_lock(Master::heartbeat_mutex);
                init_heartbeat = false;
                condition_heartbeat.notify_one();
            }
        }

        auto end_time = std::chrono::system_clock::now().time_since_epoch().count();

        if (end_time - current_time < 1000 * 1000)
        {
            std::unique_lock<std::mutex> heartbeat_lock(Master::heartbeat_mutex);
            sleep(1);
            condition_heartbeat.wait_for(heartbeat_lock, std::chrono::milliseconds(5 * 1000), [this] {
      return true; });
        }
    }
}
```

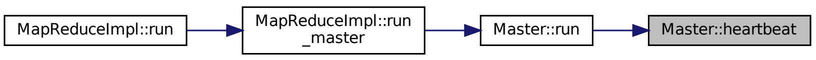
References condition_heartbeat, DEAD, handler_dead_worker(), heartbeat_mutex, heartbeat_payload::id, init_↩heartbeat, server_state, and workers.

Referenced by run().

Here is the call graph for this function:


Master::heartbeat
Master::handler_dead_worker
Master::cleanup_files
Master::find_worker_by_status
Master::find_worker_by_name

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.

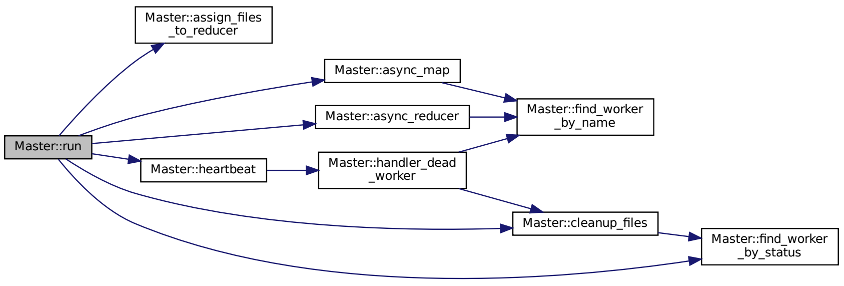
Definition at line 391 of file master.h.

```
{
    std::thread check_heartbeat_status(&Master::heartbeat, this);
#if __cplusplus >= 201703L
    fs::create_directory(TEMP_DIR);
    if (fs::is_directory(Master::mr_spec.output_directory))
    {
        for (const auto& fi : fs::directory_iterator(Master::mr_spec.output_directory))
        {
            fs::remove(fi.path());
        }
    }
#else
    auto dir_string = std::string ("rm -rf ")+ std::string( TEMP_DIR);
    system(dir_string.c_str());
    mkdir(TEMP_DIR, 0755);
#endif

    std::thread map_job(&Master::async_map, this);
    {
        std::unique_lock<std::mutex> lock_heartbeat(heartbeat_mutex);
        Master::init_heartbeat = true;
        condition_heartbeat.wait(lock_heartbeat, [this] { return !this->init_heartbeat; });
    }
    bool shards_done = false;
    Master::completion_count = Master::assigned_shards = Master::file_shards.size();
    while (!shards_done)
    {
        for (const auto& s : Master::file_shards)
        {
            int i;
            {
                std::unique_lock<std::mutex> shards(Master::cleanup_mutex);
                // wait for cleanup mutex and free worker.
                condition_cleanup_mutex.wait(shards, [this] { return
   !Master::find_worker_by_status(FREE).empty(); });
                i = Master::find_worker_by_status(FREE)[0];
                if (Master::workers[i].workerStatus == DEAD)
                {
                    continue;
                }
                Master::workers[i].current_shard = s;
                Master::workers[i].workerStatus = BUSY;
                Master::assigned_shards--;
            }
            auto client = Master::workers[i].client.get();
            //  Adds Mapper Job
            std::cout « "Assigning Map Work of shard id " + std::to_string(s.shard_id) + " to " +
                                 Master::workers[i].worker_address
                          « std::endl;

            client->schedule_mapper_jobs(Master::mr_spec, Master::workers[i].current_shard);
        }
        {
            std::unique_lock<std::mutex> work_done(ops_mutex);
            condition_ops_mutex.wait(work_done);
            if (assigned_shards <= 0 && ops_completed)
                shards_done = true;
        }

        {
            std::unique_lock<std::mutex> shards(Master::cleanup_mutex);
            if (Master::assigned_shards > 0 && !Master::missing_shards.empty())
            {
                std::cout « "Re assigning work for " « Master::missing_shards[0].shard_id « std::endl;
                Master::file_shards.clear();
                Master::file_shards.assign(Master::missing_shards.begin(),
  Master::missing_shards.end());
                Master::missing_shards.clear();
                condition_cleanup_mutex.notify_one();
            }
        }
    }
    map_job.join();
    std::cout « "Map Done." « std::endl;

    for (auto& s : Master::workers)
    {
        if (s.workerStatus == DEAD)
            continue;
        s.workerStatus = FREE;
        s.workerType = REDUCER;
    }
    ops_completed = false;

    std::thread reduce_job(&Master::async_reducer, this);

    bool partition_done = false;
    Master::completion_count = Master::assigned_partition = Master::mr_spec.output_files;
    std::vector<int> output_vector(Master::assigned_partition);
    std::iota(output_vector.begin(), output_vector.end(), 0);
    while (!partition_done && Master::assigned_partition > 0)
    {
        for (auto& i : output_vector)
        {
            int j;
            std::string output_file;
            {
                std::unique_lock<std::mutex> partition(Master::cleanup_mutex);
                condition_cleanup_mutex.wait(
                    partition, [this] { return !Master::find_worker_by_status(FREE).empty(); });
                j = Master::find_worker_by_status(FREE)[0];
                if (Master::workers[j].workerStatus == DEAD)
                {
                    continue;
                }
                Master::workers[j].workerType = REDUCER;
                output_file =
                    Master::mr_spec.output_directory + "/" +
   std::string("output_file_").append(std::to_string(i));
                Master::workers[j].output_reducer_location_map[output_file] =
   assign_files_to_reducer(i);
                Master::workers[j].current_output = i;
                Master::workers[j].workerStatus = BUSY;
                Master::assigned_partition--;
            }
            condition_cleanup_mutex.notify_one();
            auto client = Master::workers[j].client.get();
            //  Adds Reducer Job
            std::cout « "Assigning Reduce Work " + output_file + " to " +
   Master::workers[j].worker_address
                      « std::endl;

            client->schedule_reduce_job(
                Master::mr_spec, Master::workers[j].output_reducer_location_map[output_file],
   output_file);
        }
        {
            std::unique_lock<std::mutex> work_done(ops_mutex);
            condition_ops_mutex.wait(work_done);
            if (Master::assigned_partition <= 0 && ops_completed)
                partition_done = true;
        }

        {
            std::unique_lock<std::mutex> partition(Master::cleanup_mutex);
            if (Master::assigned_partition > 0 && !Master::missing_output_files.empty())
            {
                std::cout « "Re assigning work for " + Master::mr_spec.output_directory + "/" +
                                 std::string("output_file_")
                          « Master::missing_output_files[0] « std::endl;
                output_vector.clear();
                output_vector.assign(Master::missing_output_files.begin(),
   Master::missing_output_files.end());
                Master::missing_output_files.clear();
                condition_cleanup_mutex.notify_one();
            }
        }
    }

    reduce_job.join();

    {
        std::unique_lock<std::mutex> heartbeat(Master::heartbeat_mutex);
        Master::server_state = !ALIVE;
        condition_heartbeat.notify_all();
    }
    check_heartbeat_status.join();
    cleanup_files();
    return true;
}
```

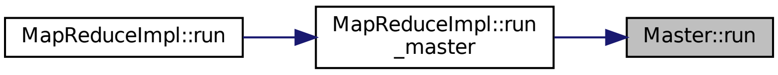
References ALIVE, assign_files_to_reducer(), assigned_partition, assigned_shards, async_map(), async_←
reducer(), BUSY, cleanup_files(), cleanup_mutex, completion_count, condition_cleanup_mutex, condition_←
heartbeat, condition_ops_mutex, DEAD, file_shards, find_worker_by_status(), FREE, heartbeat(), heartbeat_←
mutex, init_heartbeat, missing_output_files, missing_shards, mr_spec, ops_completed, ops_mutex, MapReduce←
Spec::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:


Master::assign_files _to_reducer
Master::async_map
Master::find_worker _by_name
Master::async_reducer
Master::run
Master::heartbeat
Master::handler_dead _worker
Master::cleanup_files
Master::find_worker _by_status

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().

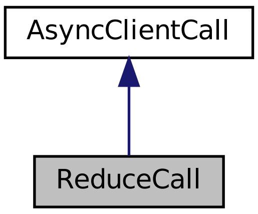
The documentation for this class was generated from the following file:

- src/master.h

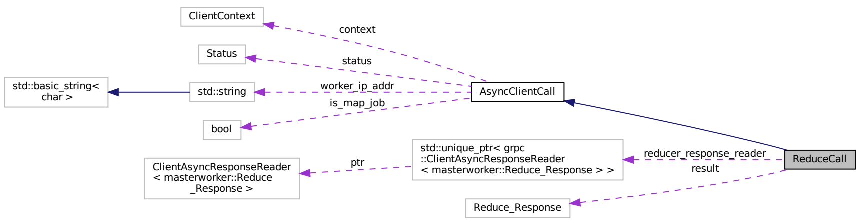
6.14 ReduceCall Class Reference

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

Inheritance diagram for ReduceCall:



Collaboration diagram for ReduceCall:



ClientContext
context
Status
status
std::basic_string< char >
std::string
worker_ip_addr
AsyncClientCall
is_map_job
bool
std::unique_ptr< grpc ::ClientAsyncResponseReader < masterworker::Reduce_Response > >
reducer_response_reader
ReduceCall
ClientAsyncResponseReader < masterworker::Reduce _Response >
ptr
result
Reduce_Response

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.

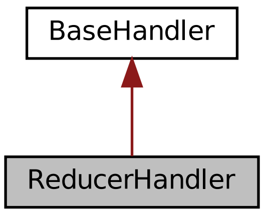
The documentation for this class was generated from the following file:

- src/master.h

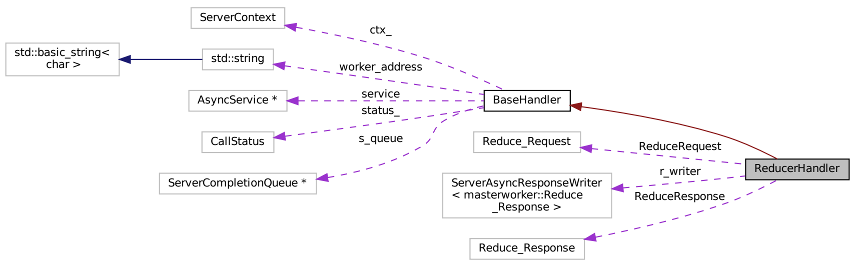
6.15 ReducerHandler Class Reference

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

Inheritance diagram for ReducerHandler:



Collaboration diagram for ReducerHandler:



Public Member Functions

- ReducerHandler (masterworker::Map_Reduce::AsyncService ∗service, grpc::ServerCompletionQueue ∗p←
Queue, std::string basicString)
- void Proceed ()

Private Member Functions

- masterworker::Reduce_Response handle_reducer_job (masterworker::Reduce_Request request)
- BaseReducerInternal ∗ get_basereducer_internal (BaseReducer ∗reducer)

Private Attributes

- masterworker::Reduce_Request ReduceRequest
- masterworker::Reduce_Response ReduceResponse
- grpc::ServerAsyncResponseWriter< masterworker::Reduce_Response > r_writer

Additional Inherited Members

6.15.1 Detailed Description

Reducer Class

Definition at line 116 of file worker.h.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 ReducerHandler()

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

Constructor for Reducer

Parameters

service	
pQueue	
basicString	

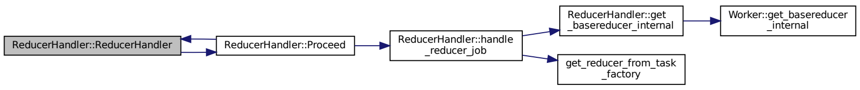
Definition at line 125 of file worker.h.

```
        : BaseHandler(service, pQueue, basicString)
        , r_writer(&ctx_)
    {
        ReducerHandler::Proceed();
    }
```

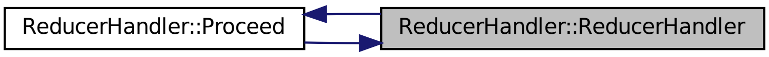
References Proceed().

Referenced by Proceed().

Here is the call graph for this function:


ReducerHandler::ReducerHandler
ReducerHandler::Proceed
ReducerHandler::handle_reducer_job
ReducerHandler::get_basereducer_internal
Worker::get_basereducer_internal
get_reducer_from_task_factory

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

reducer	

Returns

BaseReducerinternal Class

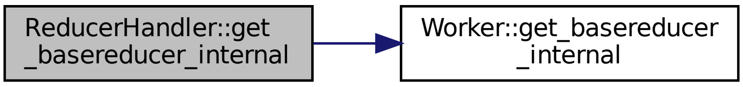
Definition at line 445 of file worker.h.

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

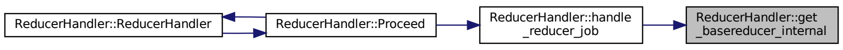
References Worker::get_basereducer_internal().

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

request	grpc Reduce Request , check .proto

Returns

grpc Reduce Response payload , check .proto

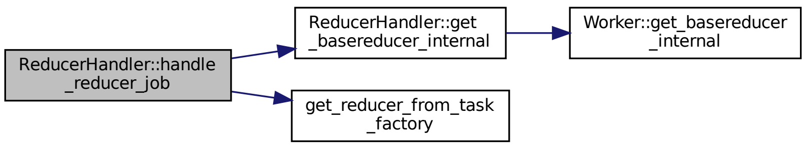
Definition at line 393 of file worker.h.

```
394 {
    masterworker::Reduce_Response payload;
    auto user_reducer_func = get_reducer_from_task_factory(request.uuid());
    auto base_reducer = get_basereducer_internal(user_reducer_func.get());
    base_reducer->file_name = request.output_file();
    std::map<std::string, std::vector<std::string» key_value_map;
    payload.set_file_name(request.output_file());
    auto d = request.file_list();
    for (const auto& f : d)
    {
        std::ifstream fs(f);
        std::string dummy;
        try{
            if (fs.good() && fs.is_open())
            {
                while (std::getline(fs, dummy))
                {
                    key_value_map[dummy.substr(0, dummy.find_first_of(DELIMITER))].push_back(
                            dummy.substr(dummy.find_first_of(DELIMITER) + 1));
                }
            }

        } catch (std::ifstream::failure &e) {
            std::cerr « f +"  Error: "+ e.what()  « std::endl;
        }
    }
    for (const auto& k : key_value_map)
    {
        user_reducer_func->reduce(k.first, k.second);
    }
    key_value_map.clear();
    return payload;
}
```

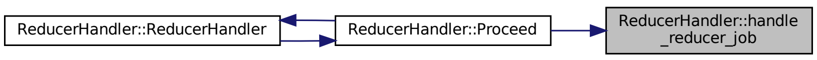
References DELIMITER, get_basereducer_internal(), and get_reducer_from_task_factory().

Referenced by Proceed().

Here is the call graph for this function:


ReducerHandler::handle_reducer_job
ReducerHandler::get_basereducer_internal
Worker::get_basereducer_internal
get_reducer_from_task_factory

Here is the caller graph for this function:


ReducerHandler::ReducerHandler
ReducerHandler::Proceed
ReducerHandler::handle_reducer_job

6.15.3.3 Proceed()

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

Reimplemented from BaseHandler.

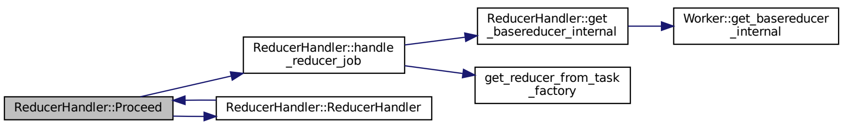
Definition at line 135 of file worker.h.

```
    {
        if (status_ == CREATE)
        {
            status_ = PROCESS;
            service->Requestreduce(&ctx_, &ReduceRequest, &r_writer, s_queue, s_queue, this);
        }
        else if (status_ == PROCESS)
        {
            new ReducerHandler(service, s_queue, worker_address);
            ReduceResponse = handle_reducer_job(ReduceRequest);
            status_ = FINISH;
            r_writer.Finish(ReduceResponse, grpc::Status::OK, this);
        }
        else
        {
            GPR_ASSERT(status_ == FINISH);
            delete this;
        }
    }
```

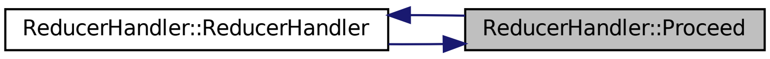
References BaseHandler::CREATE, BaseHandler::ctx_, BaseHandler::FINISH, handle_reducer_job(), BaseHandler::PROCESS, r_writer, ReduceRequest, ReduceResponse, ReducerHandler(), BaseHandler::s_queue, BaseHandler::service, BaseHandler::status_, and BaseHandler::worker_address.

Referenced by ReducerHandler().

Here is the call graph for this function:


ReducerHandler::Proceed
ReducerHandler::handle_reducer_job
ReducerHandler::ReducerHandler
ReducerHandler::get_basereducer_internal
get_reducer_from_task_factory
Worker::get_basereducer_internal

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().

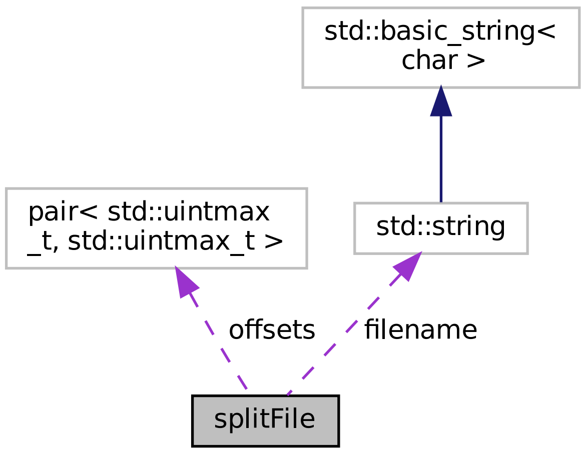
The documentation for this class was generated from the following file:

- src/worker.h

6.16 splitFile Struct Reference

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

Collaboration diagram for splitFile:


std::basic_string< char >
pair< std::uintmax _t, std::uintmax_t >
std::string
offsets
filename
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.

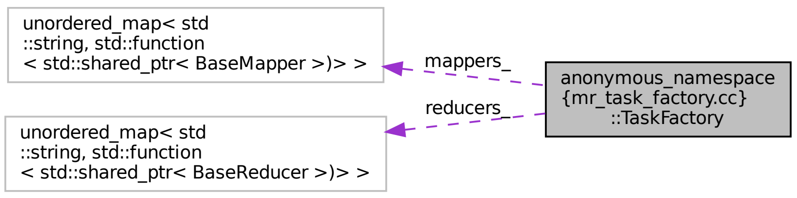
Referenced by shard_files().

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:


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

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.

```
                                                                                 {
        auto itr = mappers_.find(user_id);
        if (itr == mappers_.end())
            return nullptr;
        return itr->second();
    }
```

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.

```
                                                                          {
        auto itr = reducers_.find(user_id);
        if (itr == reducers_.end())
            return nullptr;
        return itr->second();
    }
```

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.

```
                                    {

        static TaskFactory *instance = new TaskFactory();
        return *instance;
    }
```

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.

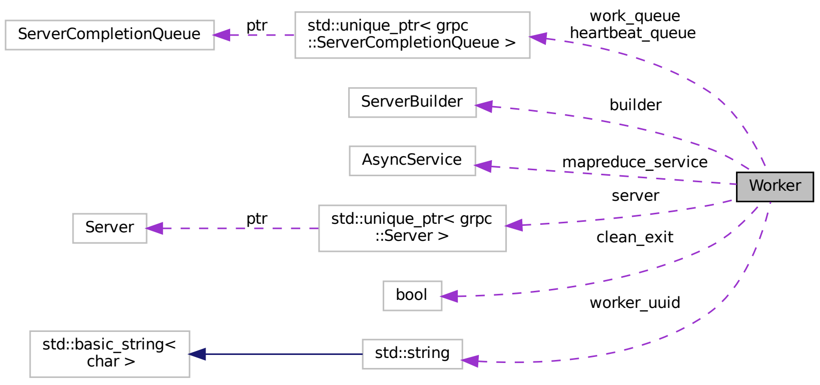
The documentation for this class was generated from the following file:

- 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

ip_addr_port	

Definition at line 265 of file worker.h.

```
{
    std::cout « "listening on " « ip_addr_port « std::endl;
    Worker::builder.AddListeningPort(ip_addr_port, grpc::InsecureServerCredentials());
    Worker::builder.RegisterService(&this->mapreduce_service);
    Worker::work_queue = Worker::builder.AddCompletionQueue();
    Worker::heartbeat_queue = Worker::builder.AddCompletionQueue();
    Worker::worker_uuid = ip_addr_port.substr(ip_addr_port.find_first_of(':') + 1);
}
```

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.

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

References clean_exit, and server.

6.18.3 Member Function Documentation

6.18.3.1 get_basemapper_internal()

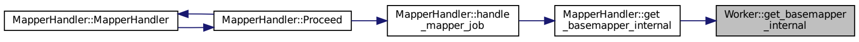
```
static BaseMapperInternal* Worker::get_basemapper_internal (
            BaseMapper * mapper ) [inline], [static]
```

Definition at line 240 of file worker.h.

```
    {
        return mapper->impl_;
    }
```

Referenced by MapperHandler::get_basemapper_internal().

Here is the caller graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle_mapper_job
MapperHandler::get_basemapper_internal
Worker::get_basemapper_internal

6.18.3.2 get_basereducer_internal()

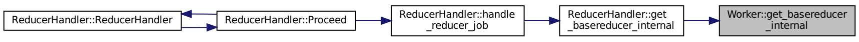
```
static BaseReducerInternal* Worker::get_basereducer_internal (
            BaseReducer * reducer ) [inline], [static]
```

Definition at line 235 of file worker.h.

```
    {
        return reducer->impl_;
    }
```

Referenced by ReducerHandler::get_basereducer_internal().

Here is the caller graph for this function:


ReducerHandler::ReducerHandler
ReducerHandler::Proceed
ReducerHandler::handle_reducer_job
ReducerHandler::get_basereducer_internal
Worker::get_basereducer_internal

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.

```
{
    void* tag;
    bool ok;

    new HeartbeatHandler(&(Worker::mapreduce_service), heartbeat_queue.get(), worker_uuid);

    while (true)
    {
        if (Worker::clean_exit)
            return;
        GPR_ASSERT(heartbeat_queue->Next(&tag, &ok));

        static_cast<BaseHandler*>(tag)->Proceed();
    }
}
```

References clean_exit, heartbeat_queue, mapreduce_service, and worker_uuid.

Referenced by run().

Here is the caller graph for this function:

Worker::run
Worker::heartbeat_handler

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 BaseReduer's member BaseReducerInternal impl_ directly, so you can manipulate them however you want when running map/reduce tasks

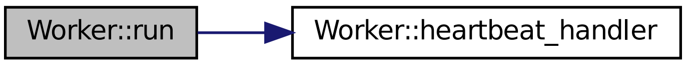
Returns

Definition at line 284 of file worker.h.

```
{
    void* tag;
    bool ok;
    Worker::server = Worker::builder.BuildAndStart();
    std::thread heartbeat_job(&Worker::heartbeat_handler, this);
    new MapperHandler(&(Worker::mapreduce_service), work_queue.get(), worker_uuid);
    new ReducerHandler(&(Worker::mapreduce_service), work_queue.get(), worker_uuid);

    while (true)
    {
        GPR_ASSERT(work_queue->Next(&tag, &ok));
        static_cast<BaseHandler*>(tag)->Proceed();
    }
    return true;
}
```

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().

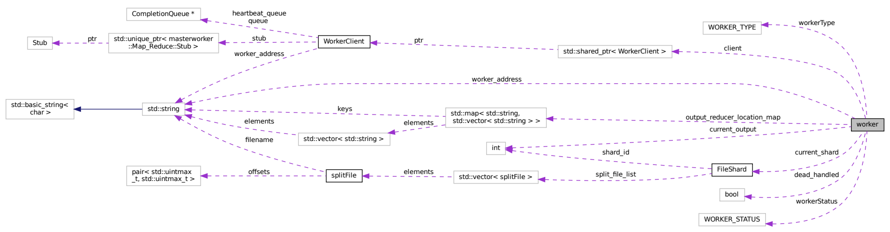
The documentation for this class was generated from the following file:

- 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().

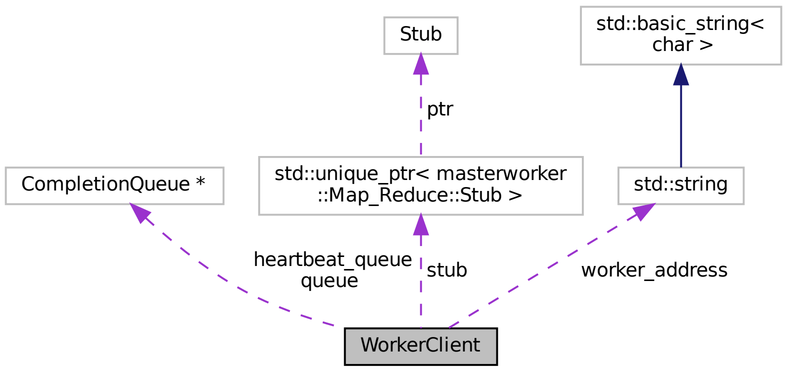
The documentation for this struct was generated from the following file:

- 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

address	
queue	

Definition at line 119 of file master.h.

```
    : queue(queue)
    , worker_address(address)
{
    std::cout « "creating channel at " + address « std::endl;
    heartbeat_queue = new grpc::CompletionQueue();
    this->stub = masterworker::Map_Reduce::NewStub(grpc::CreateChannel(address,
   grpc::InsecureChannelCredentials()));
}
```

References heartbeat_queue, and stub.

6.20.2.2 ∼WorkerClient()

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

Definition at line 101 of file master.h.

```
    {
        heartbeat_queue->Shutdown();
    }
```

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

shard	
partition	

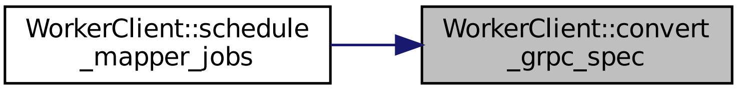
Definition at line 206 of file master.h.

```
{
    partition->set_shard_id(shard->shard_id);
    for (auto f : shard->split_file_list)
    {
        auto temp = partition->add_file_list();
        temp->set_filename(f.filename);
        temp->set_start_offset(f.offsets.first);
        temp->set_end_offset(f.offsets.second);
    }
}
```

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.

```
{
    void* tag;
    bool ok = false;
    GPR_ASSERT(WorkerClient::heartbeat_queue->Next(&tag, &ok));
    auto* call = static_cast<HeartbeatCall*>(tag);
    if (call->status.ok())
    {
        if (call->result.status() == masterworker::Heartbeat_Payload_type_DEAD)
        {
            std::cerr « "Error " « call->worker_ip_addr « " : Dead" « std::endl;
            return false;
        }
        delete call;
        return true;
    }
    auto temp = call->status;
    std::cerr « "Error " « this->worker_address « " : " « call->status.error_message()
              « "details : " « call->status.error_details() « "status code " « call->status.error_code()
      « "   ok "
              « call->status.ok() « std::endl;
    return false;
}
```

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 )
```

Sechules mapper job with Worker Client, similar to reduce , heartbeat etc

Parameters

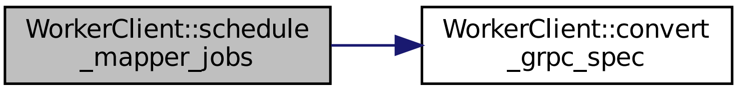
spec	
shard	

Definition at line 223 of file master.h.

```
{
    masterworker::Map_Request mapRequest;
    mapRequest.set_uuid(spec.user);
    mapRequest.set_partition_count(spec.output_files);
    auto s = mapRequest.add_shard();
    this->convert_grpc_spec(&shard, s);
    auto call = new MapCall;
    call->worker_ip_addr = WorkerClient::worker_address;
    call->map_response_reader = WorkerClient::stub->PrepareAsyncmap(&call->context, mapRequest,
   WorkerClient::queue);
    call->is_map_job = true;
    call->map_response_reader->StartCall();
    call->map_response_reader->Finish(&call->result, &call->status, (void*)call);
}
```

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:


WorkerClient::schedule_mapper_jobs
WorkerClient::convert_grpc_spec

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

spec	mapreduce ini file data structure.
file_list	
output_file_location	

Definition at line 181 of file master.h.

```
{
    masterworker::Reduce_Request reduceRequest;
    reduceRequest.set_uuid(spec.user);
    reduceRequest.set_output_file(output_file_location);
    for (const auto& l : file_list)
    {
        auto f = reduceRequest.add_file_list();
        f->append(l);
    }
    auto call = new ReduceCall;
    call->worker_ip_addr = this->worker_address;
    call->reducer_response_reader = WorkerClient::stub->PrepareAsyncreduce(&call->context,
   reduceRequest, WorkerClient::queue);
    call->is_map_job = false;
    call->reducer_response_reader->StartCall();
    call->reducer_response_reader->Finish(&call->result, &call->status, (void*)call);
}
```

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

current_time	

Definition at line 132 of file master.h.

```
{
    //    std::cout « "Info " « std::chrono::system_clock::to_time_t(std::chrono::system_clock::now()) «
   " " +
    //    this->worker_address « " : Hbeat sent" « std::endl;

    std::chrono::system_clock::time_point deadline = std::chrono::system_clock::now() +
   std::chrono::seconds(TIMEOUT);
    auto call = new HeartbeatCall;
    call->worker_ip_addr = this->worker_address;
    call->context.set_deadline(deadline);
    masterworker::Heartbeat_Payload payload;
    payload.set_id(this->worker_address);
    //    payload.set_timestamp(current_time);
    payload.set_status(masterworker::Heartbeat_Payload_type_UNKNOWN);
    call->heartbeat_payload_reader = WorkerClient::stub->PrepareAsyncheartbeat(&call->context, payload,
   WorkerClient::heartbeat_queue);
    call->heartbeat_payload_reader->StartCall();
    call->heartbeat_payload_reader->Finish(&call->result, &call->status, (void*)call);
}
```

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 )
```

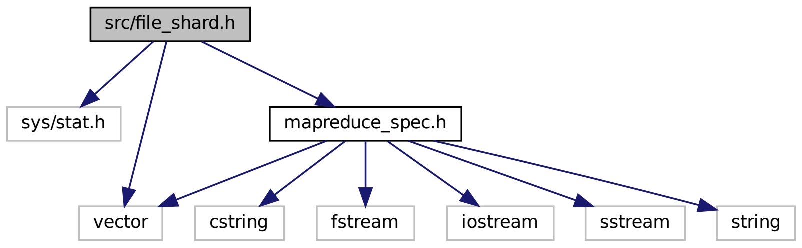
Definition at line 2 of file CMakeLists.txt.

```
19                                          {MAPREDUCE_INCLUDE_DIR})
```

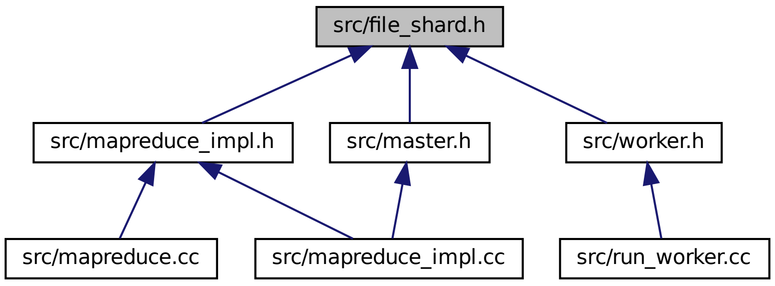
7.2 src/file_shard.h File Reference

```
#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

fileName	
offset	
optimal_shard_size	

Returns

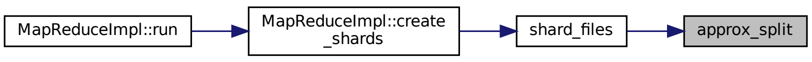
nearest file Offset for `\n` from given file offset. usually used as offset + return_value

Definition at line 46 of file file_shard.h.

```
{
    std::uintmax_t approx_size;
    std::ifstream fs(fileName);
    if (!fs.good())
    {
        std::cerr « "Error Opening file: " « fileName « std::endl;
        return 0;
    }
    fs.seekg(offset + optimal_shard_size);
    std::string temp_str;
    std::getline(fs, temp_str);
    approx_size = optimal_shard_size + temp_str.length() + 1;
    return approx_size;
}
```

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

path	

Returns

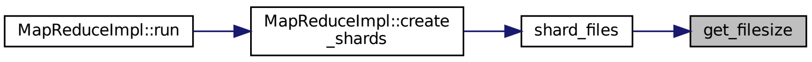
file size usually with 64bit value

Definition at line 14 of file file_shard.h.

```
{
#if __cplusplus >= 201703L
    return fs::file_size(path);
#else
    struct stat stat_buf;
    int rc = stat(path.c_str(), &stat_buf);
    return rc == 0 ? stat_buf.st_size : -1;
#endif
}
```

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

mr_spec	
fileShards	

Returns

true if succeeded.

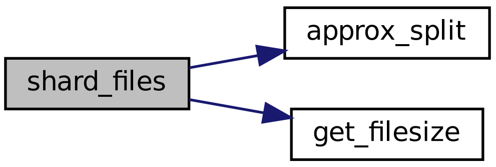
Definition at line 72 of file file_shard.h.

```
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,
       rem_shard_size);
99                 current_split_file.offsets = {offset, offset + nearest_size};
100                 if (offset > offset + nearest_size)
101                 {
102                     perror("SOMETHING WENT WRONG......");
103                     exit(1);
104                 }
105                 current_shard.split_file_list.push_back(current_split_file);
106                 current_split_file = splitFile();
107                 rem_shard_size -= nearest_size;
108                 rem_file_size -= nearest_size;
109                 offset += nearest_size;
110             }
111             if (rem_shard_size <= 0)
112             {
113                 fileShards.push_back(current_shard);
114                 current_shard = FileShard();
115                 current_shard.shard_id = fileShards.size();
116                 rem_shard_size = optimal_shard_size;
117             }
118         }
119     }
120     if (current_shard.shard_id > -1)
121         fileShards.push_back(current_shard);
122     return true;
123 }
```

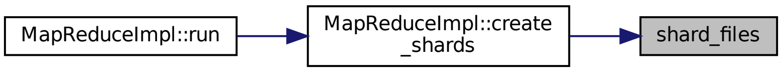
References approx_split(), splitFile::filename, get_filesize(), MapReduceSpec::input_files, KB, MapReduceSpec↩ ::map_kb, splitFile::offsets, FileShard::shard_id, and FileShard::split_file_list.

Referenced by MapReduceImpl::create_shards().

Here is the call graph for this function:



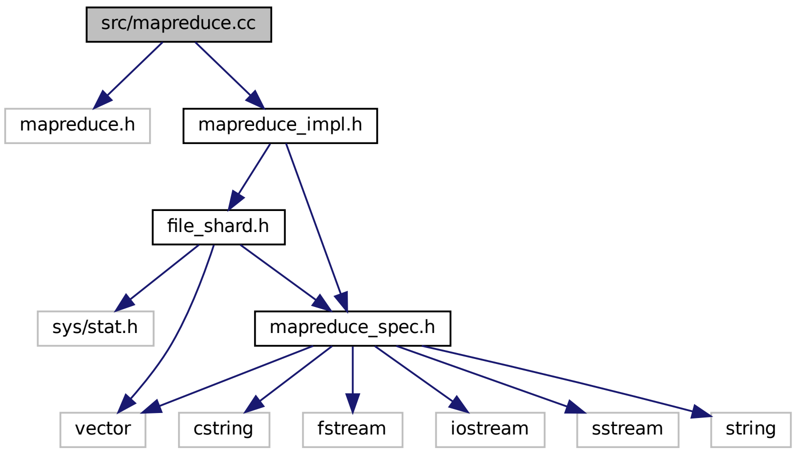
Here is the caller graph for this function:



7.3 src/mapreduce.cc File Reference

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

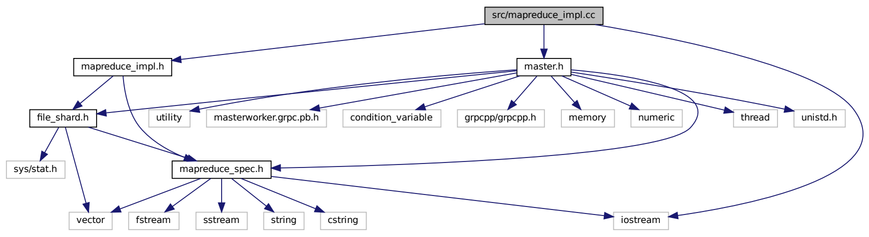
Include dependency graph for mapreduce.cc:



7.4 src/mapreduce_impl.cc File Reference

```
#include "mapreduce_impl.h"
#include <iostream>
#include "master.h"
```

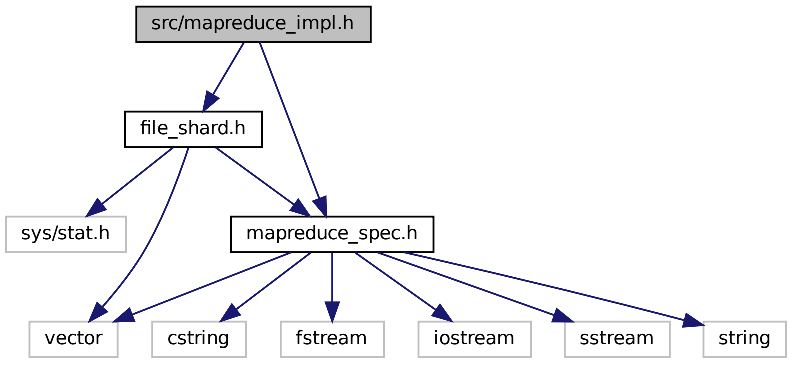
Include dependency graph for mapreduce_impl.cc:



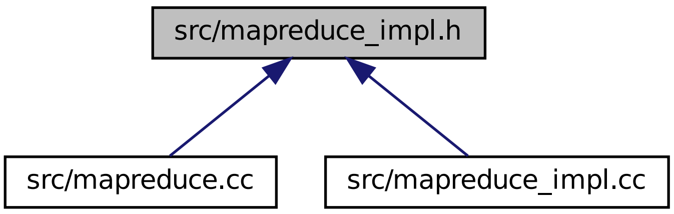
7.5 src/mapreduce_impl.h File Reference

```
#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:



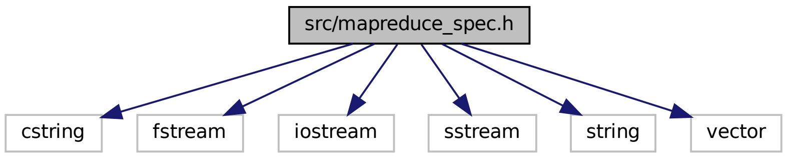
Data Structures

- class MapReduceImpl

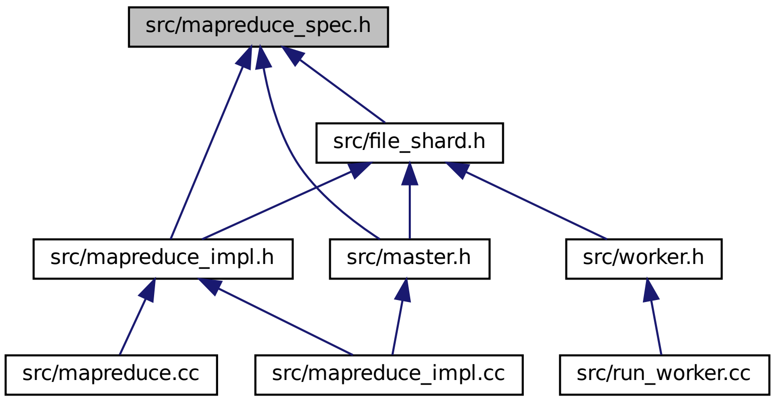
7.6 src/mapreduce_spec.h File Reference

```
#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

config_filename	
mr_spec	

Returns

true or false based on success

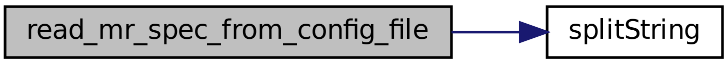
Definition at line 59 of file mapreduce_spec.h.

```
{
    std::ifstream config_file(config_filename);
    std::string config_line;
    if (!config_file.good())
    {
        std::cerr « "Error opening fie : " « config_filename « " Error No" « std::strerror(errno) « std::endl;
        return false;
    }
    while (std::getline(config_file, config_line))
    {
        std::string key, value;
        key = config_line.substr(0, config_line.find_first_of('='));
        value = config_line.substr(config_line.find_first_of('=') + 1, config_line.length());
        if (value.empty() || key.empty())
        {
            std::cerr « key « " is empty or value " « value « " is empty , please check again" « std::endl;
            return false;
        }
        if (key == "n_workers")
        {
            mr_spec.worker_count = std::stoi(value);
            continue;
        }
        if (key == "worker_ipaddr_ports")
        {
            mr_spec.worker_endpoints = splitString(value, ',');
            continue;
        }
        if (key == "input_files")
        {
            mr_spec.input_files = splitString(value, ',');

            continue;
        }
        if (key == "output_dir")
        {
            mr_spec.output_directory = value;
            continue;
        }
        if (key == "n_output_files")
        {
            mr_spec.output_files = std::stoi(value);
            continue;
        }
        if (key == "map_kilobytes")
        {
            mr_spec.map_kb = std::stoi(value);
            continue;
        }
        if (key == "user_id")
        {
            mr_spec.user = value;
            continue;
        }
    }

    return true;
}
```

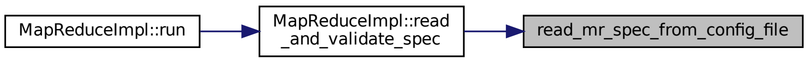
References MapReduceSpec::input_files, MapReduceSpec::map_kb, MapReduceSpec::output_directory, MapReduceSpec::output_files, splitString(), MapReduceSpec::user, MapReduceSpec::worker_count, and MapReduceSpec::worker_endpoints.

Referenced by MapReduceImpl::read_and_validate_spec().

Here is the call graph for this function:



Here is the caller graph for this function:


MapReduceImpl::run
MapReduceImpl::read_and_validate_spec
read_mr_spec_from_config_file

7.6.1.2 splitString()

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

Splits string with given delimiter del

Parameters

s	--> raw string
del	--> delimiter

Returns

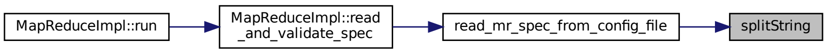
returns vector with parsed strings.

Definition at line 40 of file mapreduce_spec.h.

```
{
    std::vector<std::string> arr{};
    std::stringstream ss(s);
    std::string temp;
    while (std::getline(ss, temp, del))
    {
        arr.push_back(temp);
    }
    return arr;
}
```

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

mr_spec	

Returns

true or false based on validation criteria.

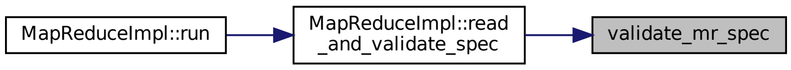
Definition at line 125 of file mapreduce_spec.h.

```
{

    if (mr_spec.worker_endpoints.size() != mr_spec.worker_count)
    {
        std::cerr « "Invalid Count of Workers : " « mr_spec.worker_endpoints.size() « "config -
      worker_count"
                  « mr_spec.worker_count « std::endl;
        return false;
    }

#if __cplusplus >= 201703L
    if (!fs::is_directory(mr_spec.output_directory))
    {
        if (fs::is_regular_file(mr_spec.output_directory))
        {
            std::cerr « mr_spec.output_directory « " is file not directory please provide correct path."
      « std::endl;
            return false;
        }
        else
        {
            try
            {
                fs::create_directory(mr_spec.output_directory);
            }
            catch (fs::filesystem_error& e)
            {
                std::cout « e.what() « std::endl;
            }
        }
    }
#endif
    for (const auto& f : mr_spec.input_files)
    {
        std::ifstream temp_stream(f);
        if (!temp_stream.good())
        {
            std::cerr « "Error opening fie : " « f « " Error No: " « std::strerror(errno) « std::endl;
            return false;
        }
    }
    return true;
}
```

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

Referenced by MapReduceImpl::read_and_validate_spec().

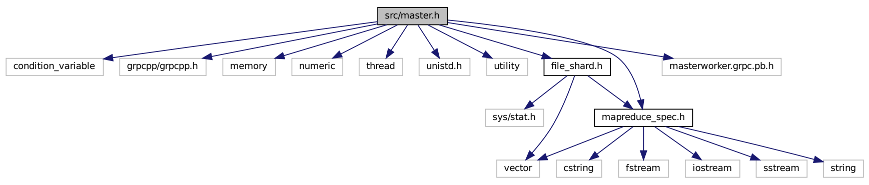
Here is the caller graph for this function:



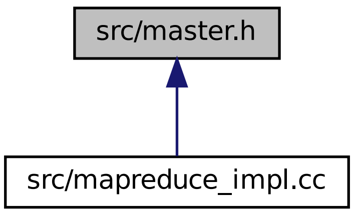
7.7 src/master.h File Reference

```
#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.

```
{
    FREE,
    BUSY,
    DEAD
};
```

7.7.2.2 WORKER_TYPE

`enum WORKER_TYPE`

Enumerator

MAPPER	
REDUCER	

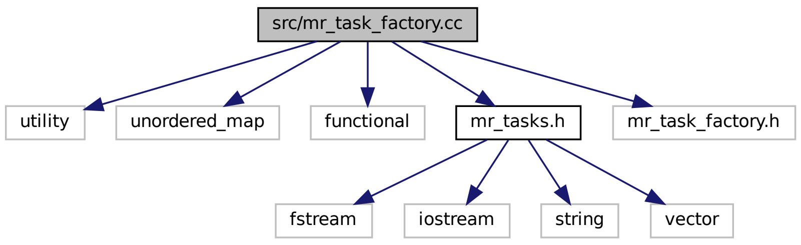
Definition at line 32 of file master.h.

```
{
    MAPPER,
    REDUCER
};
```

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()

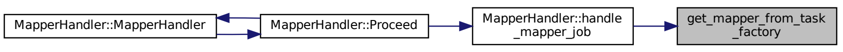
```
std::shared_ptr<BaseMapper> get_mapper_from_task_factory (
            const std::string & user_id )
```

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()

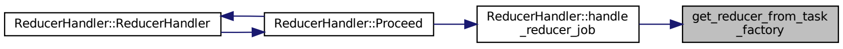
```
std::shared_ptr<BaseReducer> get_reducer_from_task_factory (
            const std::string & user_id )
```

Definition at line 86 of file mr_task_factory.cc.

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

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 )
```

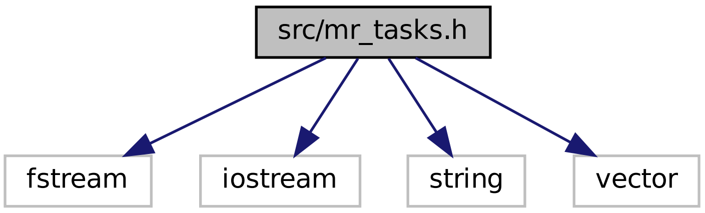
Definition at line 74 of file mr_task_factory.cc.

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

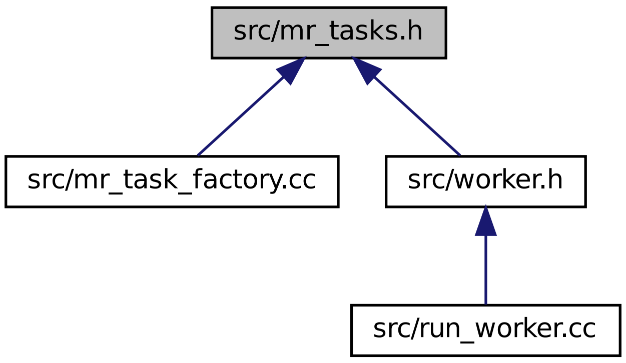
7.9 src/mr_tasks.h File Reference

```
#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
```

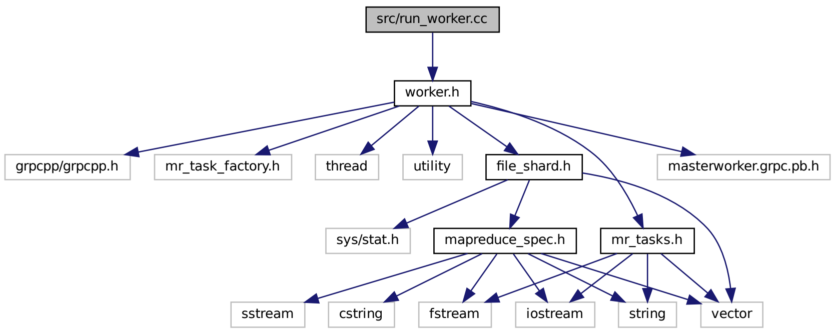
Definition at line 10 of file mr_tasks.h.

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 )
```

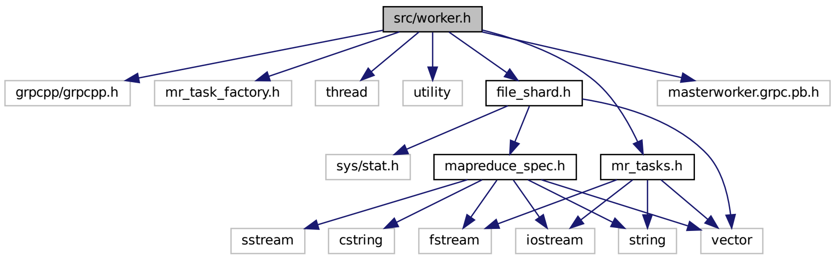
Definition at line 3 of file run_worker.cc.

```
{
    std::string ip_addr_port;
    if (argc == 2)
    {
        ip_addr_port = std::string(argv[1]);
    }
     else
     {
         std::cerr « "Correct usage: [$binary_name $ip_addr_port], example: [./mr_worker localhost:50051]"
     « std::endl;
         return EXIT_FAILURE;
     }

     Worker worker(ip_addr_port);
     return worker.run() ? EXIT_SUCCESS : EXIT_FAILURE;
 }
```

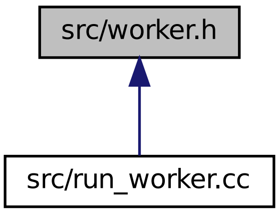
7.12 src/worker.h File Reference

```
#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()

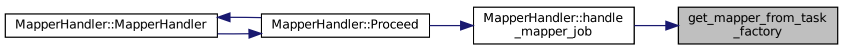
```
std::shared_ptr<BaseMapper> get_mapper_from_task_factory (
            const std::string & user_id )
```

Definition at line 81 of file mr_task_factory.cc.

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

Referenced by MapperHandler::handle_mapper_job().

Here is the caller graph for this function:


MapperHandler::MapperHandler
MapperHandler::Proceed
MapperHandler::handle _mapper_job
get_mapper_from_task _factory

7.12.1.2 get_reducer_from_task_factory()

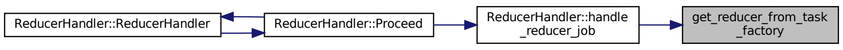
```
std::shared_ptr<BaseReducer> get_reducer_from_task_factory (
            const std::string & user_id )
```

Definition at line 86 of file mr_task_factory.cc.

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

Referenced by ReducerHandler::handle_reducer_job().

Here is the caller graph for this function:


ReducerHandler::ReducerHandler
ReducerHandler::Proceed
ReducerHandler::handle_reducer_job
get_reducer_from_task_factory

Index