# 代码导出pdf测试

```
#include "HelloWorldPubSubTypes.h"  //自定义数据生成的序列化和反序列化文件

#include <fastdds/dds/domain/DomainParticipantFactory.hpp>  //用于创建和销毁DomainParticipant对象
#include <fastdds/dds/domain/DomainParticipant.hpp>
#include <fastdds/dds/topic/TypeSupport.hpp>
#include <fastdds/dds/publisher/Publisher.hpp>
#include <fastdds/dds/publisher/DataWriter.hpp>
#include <fastdds/dds/publisher/DataWriterListener.hpp>
using namespace eprosima::fastdds::dds;         //名称空间

class HelloWorldPublisher                       //发布者类
{
public:
    HelloWorldPublisher(): participant_(nullptr), publisher_(nullptr),
                           topic_(nullptr), writer_(nullptr),
                           type_(new HelloWorldPubSubType()){}

    virtual ~HelloWorldPublisher()
    {   //手动析构，清空指针成员
        if (writer_ != nullptr){
            publisher_->delete_datawriter(writer_);
        }
        if (publisher_ != nullptr){
            participant_->delete_publisher(publisher_);
        }
        if (topic_ != nullptr){
            participant_->delete_topic(topic_);
        }
        DomainParticipantFactory::get_instance()->delete_participant(participant_);
    }

    //初始化发布者
    bool init()
    {
        hello_.index(0);                               //要发布的数据
        hello_.message("HelloWorld");

        //实例化participant_
        DomainParticipantQos participantQos;            //要为参与者设置的QOS
        participantQos.name("Participant_publisher");
        participant_ = DomainParticipantFactory::get_instance()->create_parti
```

```
cipant(0, participantQos);
        if (participant_ == nullptr) return false;

        //向参与者注册话题数据类型
        type_.register_type(participant_);

        //由参与者构造发布话题
        topic_ = participant_->create_topic("HelloWorldTopic", "HelloWorld", TOPIC_QOS_DEFAULT);
        if (topic_ == nullptr)  return false;

        //由参与者构造发布者
        publisher_ = participant_->create_publisher(PUBLISHER_QOS_DEFAULT, nullptr);
        if (publisher_ == nullptr)  return false;

        //由发布者构造DataWriter
        writer_ = publisher_->create_datawriter(topic_, DATAWRITER_QOS_DEFAULT, &listener_);
        if (writer_ == nullptr)  return false;

        return true;
    }

    //执行发布动作
    bool publish()
    {
        if (listener_.matched_ > 0)    //当出现匹配的订阅者时进行发布
        {
            hello_.index(hello_.index() + 1);
            writer_->write(&hello_);
            return true;
        }
        return false;
    }

    //发布函数接口
    void run(uint32_t samples)
    {
        uint32_t samples_sent = 0;
        while (samples_sent < samples)  //控制发布一定的次数
        {
            if (publish()){
                samples_sent++;
                std::cout << "Message: " << hello_.message() << " with index: " << hello_.index() << " SENT" << std::endl;
            }
            std::this_thread::sleep_for(std::chrono::milliseconds(1000));
//控制发布频率
        }
    }

private:
    HelloWorld hello_;                      //idl文件自定义数据的类对象
    DomainParticipant* participant_;        //参与者(类似于ros中的node),用于构建发布者、订阅者和话题
    Publisher* publisher_;                  //发布者，负责创建DataWriters的对象
    Topic* topic_;                          //话题
    DataWriter* writer_;                    //向发布话题中写入数据
    TypeSupport type_;                      //为参与者提供序列化、反序列化和获取特定数据类型的密钥的功能。

    class PubListener : public DataWriterListener  //继承DataWriterListener
    {
    public:
        PubListener(): matched_(0){}
        ~PubListener() override{}

        //覆盖默认的DataWriter侦听器回调。当检测到新的DataReader侦听DataWriter发布的话题时，重写回调函数来自定义一些操作
        void on_publication_matched(DataWriter*,const PublicationMatchedStatus& info) override
        {
            if (info.current_count_change == 1){            //发布/订阅的匹配数发生改变，多了一组
                matched_ = info.total_count;                //注意这里用的是历史累计匹配数，不是当前匹配数
                std::cout << "Publisher matched." << std::endl;
            }else if (info.current_count_change == -1){     //发布/订阅的匹配数发生改变，少了一组
                matched_ = info.total_count;
                std::cout << "Publisher unmatched." << std::endl;
            }else{
                std::cout << info.current_count_change << " is not a valid value for PublicationMatchedStatus current count change." << std::endl;
            }
        }
        std::atomic_int matched_;                           //累计匹配的订阅者个数
    } listener_;                            //listener_对象是为了跟踪订阅状态的变化
};

int main(int argc,char** argv)
{
    std::cout << "Starting publisher." << std::endl;
```

```cpp
    int samples = 10;                       //控制发布10次信息

    HelloWorldPublisher* mypub = new HelloWorldPublisher();     //发布对象的指针
    if(mypub->init()){                       //初始化
        mypub->run(static_cast<uint32_t>(samples)); //执行具体的发布
    }

    delete mypub;                           //清空指针后退出
    return 0;
}
```

```cmake
cmake_minimum_required(VERSION 3.2)      #要求最小版本3.2,和上面有些不同

project(cartographer)                    #工程名

set(CARTOGRAPHER_MAJOR_VERSION 1)        #设置版本号
set(CARTOGRAPHER_MINOR_VERSION 0)
set(CARTOGRAPHER_PATCH_VERSION 0)
set(CARTOGRAPHER_VERSION ${CARTOGRAPHER_MAJOR_VERSION}.${CARTOGRAPHER_MINOR_VERSION}.${CARTOGRAPHER_PATCH_VERSION})
set(CARTOGRAPHER_SOVERSION ${CARTOGRAPHER_MAJOR_VERSION}.${CARTOGRAPHER_MINOR_VERSION})
option(BUILD_GRPC "build Cartographer gRPC support" false) #是否构建GRPC，false
set(CARTOGRAPHER_HAS_GRPC ${BUILD_GRPC})
option(BUILD_PROMETHEUS "build Prometheus monitoring support" false)

include("${PROJECT_SOURCE_DIR}/cmake/functions.cmake")    #引入functions.cmake文件
google_initialize_cartographer_project()    #执行函数(引入文件中定义的函数)
# google_enable_testing()

find_package(absl REQUIRED)
set(BOOST_COMPONENTS iostreams)
if(WIN32)
  list(APPEND BOOST_COMPONENTS zlib)
  set(Boost_USE_STATIC_LIBS FALSE)
endif()
find_package(Boost REQUIRED COMPONENTS ${BOOST_COMPONENTS})
find_package(Ceres REQUIRED COMPONENTS SuiteSparse)
find_package(Eigen3 REQUIRED)
find_package(LuaGoogle REQUIRED)
if(WIN32)
```

```cmake
  # On Windows, Protobuf is incorrectly found by the bundled CMake module, so prefer native CMake config.
  set(protobuf_MODULE_COMPATIBLE TRUE CACHE INTERNAL "")
  find_package(Protobuf 3.0.0 CONFIG)
else()
  find_package(Protobuf 3.0.0 REQUIRED)
endif()

if (${BUILD_GRPC})
  find_package(async_grpc REQUIRED)
endif()

if(${BUILD_PROMETHEUS})
  find_package( ZLIB REQUIRED )
endif()

include(FindPkgConfig)
if (NOT WIN32)
  PKG_SEARCH_MODULE(CAIRO REQUIRED cairo>=1.12.16)
else()
  find_library(CAIRO_LIBRARIES cairo)
endif()

# Only build the documentation if we can find Sphinx.
find_package(Sphinx)
if(SPHINX_FOUND)
  add_subdirectory("docs")
endif()

#安装package.xml
install(FILES package.xml DESTINATION share/cartographer)

set(CARTOGRAPHER_CONFIGURATION_FILES_DIRECTORY ${CMAKE_INSTALL_PREFIX}/share/cartographer/configuration_files
  CACHE PATH ".lua configuration files directory")

install(DIRECTORY configuration_files DESTINATION share/cartographer/)    #安装配置文件夹

install(DIRECTORY cmake DESTINATION share/cartographer/)

file(GLOB_RECURSE ALL_LIBRARY_HDRS "cartographer/*.h")    #找到目录下所有.h文件，放到变量ALL_LIBRARY_HDRS中
file(GLOB_RECURSE ALL_LIBRARY_SRCS "cartographer/*.cc")
file(GLOB_RECURSE TEST_LIBRARY_HDRS "cartographer/fake_*.h" "cartographer/*test_helpers*.h" "cartographer/mock_*.h")
file(GLOB_RECURSE TEST_LIBRARY_SRCS "cartographer/fake_*.cc" "cartographer/*test_helpers*.cc" "cartographer/mock_*.cc")
file(GLOB_RECURSE ALL_TESTS "cartographer/*_test.cc")
file(GLOB_RECURSE ALL_EXECUTABLES "cartographer/*_main.cc")

# Remove dotfiles/-folders that could potentially pollute the build.
# 移除所有以.开头的文件
file(GLOB_RECURSE ALL_DOTFILES ".*/*")
if (ALL_DOTFILES)
  list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_DOTFILES})
  list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_DOTFILES})
  list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_DOTFILES})
  list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_DOTFILES})
  list(REMOVE_ITEM ALL_TESTS ${ALL_DOTFILES})
  list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_DOTFILES})
endif()

list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_EXECUTABLES})
list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_TESTS})
list(REMOVE_ITEM ALL_LIBRARY_HDRS ${TEST_LIBRARY_HDRS})
list(REMOVE_ITEM ALL_LIBRARY_SRCS ${TEST_LIBRARY_SRCS})
file(GLOB_RECURSE ALL_GRPC_FILES "cartographer/cloud/*")
file(GLOB_RECURSE ALL_PROMETHEUS_FILES "cartographer/cloud/metrics/prometheus/*")
list(REMOVE_ITEM ALL_GRPC_FILES ${ALL_PROMETHEUS_FILES})
if (NOT ${BUILD_GRPC})
  list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_GRPC_FILES})
  list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_GRPC_FILES})
  list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_GRPC_FILES})
  list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_GRPC_FILES})
  list(REMOVE_ITEM ALL_TESTS ${ALL_GRPC_FILES})
  list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_GRPC_FILES})
endif()
if (NOT ${BUILD_PROMETHEUS})
  list(REMOVE_ITEM ALL_LIBRARY_HDRS ${ALL_PROMETHEUS_FILES})
  list(REMOVE_ITEM ALL_LIBRARY_SRCS ${ALL_PROMETHEUS_FILES})
  list(REMOVE_ITEM TEST_LIBRARY_HDRS ${ALL_PROMETHEUS_FILES})
  list(REMOVE_ITEM TEST_LIBRARY_SRCS ${ALL_PROMETHEUS_FILES})
  list(REMOVE_ITEM ALL_TESTS ${ALL_PROMETHEUS_FILES})
  list(REMOVE_ITEM ALL_EXECUTABLES ${ALL_PROMETHEUS_FILES})
endif()

set(INSTALL_SOURCE_HDRS ${ALL_LIBRARY_HDRS} ${TEST_LIBRARY_HDRS})

# 移除internal文件夹内的头文件,不进行安装
file(GLOB_RECURSE INTERNAL_HDRS "cartographer/*/internal/*.h")
list(REMOVE_ITEM INSTALL_SOURCE_HDRS ${INTERNAL_HDRS})

file(GLOB_RECURSE ALL_PROTOS "cartographer/*.proto")
file(GLOB_RECURSE ALL_GRPC_SERVICES "cartographer/*_service.proto")
```

```
list(REMOVE_ITEM ALL_PROTOS ALL_GRPC_SERVICES)
if (NOT ${BUILD_GRPC})
  list(REMOVE_ITEM ALL_PROTOS ${ALL_GRPC_FILES})
endif()

# TODO(cschuet): Move proto compilation to separate function.
set(ALL_PROTO_SRCS)
set(ALL_PROTO_HDRS)
foreach(ABS_FIL ${ALL_PROTOS})
  file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
  get_filename_component(DIR ${REL_FIL} DIRECTORY)
  get_filename_component(FIL_WE ${REL_FIL} NAME_WE)

  list(APPEND ALL_PROTO_SRCS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc")
  list(APPEND ALL_PROTO_HDRS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h")

  add_custom_command(
    OUTPUT "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc"
           "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h"
    COMMAND  ${PROTOBUF_PROTOC_EXECUTABLE}
    ARGS --cpp_out  ${PROJECT_BINARY_DIR} -I
      ${PROJECT_SOURCE_DIR} ${ABS_FIL}
    DEPENDS ${ABS_FIL}
    COMMENT "Running C++ protocol buffer compiler on ${ABS_FIL}"
    VERBATIM
  )
endforeach()
set_source_files_properties(${ALL_PROTO_SRCS} ${ALL_PROTO_HDRS} PROPERTIES GENERATED TRUE)

# 添加.pb.h与.pb.cc到头文件源文件列表
list(APPEND ALL_LIBRARY_HDRS ${ALL_PROTO_HDRS})
list(APPEND ALL_LIBRARY_SRCS ${ALL_PROTO_SRCS})

if(${BUILD_GRPC})
  set(ALL_GRPC_SERVICE_SRCS)
  set(ALL_GRPC_SERVICE_HDRS)
  foreach(ABS_FIL ${ALL_GRPC_SERVICES})
    file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
    get_filename_component(DIR ${REL_FIL} DIRECTORY)
    get_filename_component(FIL_WE ${REL_FIL} NAME_WE)

    list(APPEND ALL_GRPC_SERVICE_SRCS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc")
    list(APPEND ALL_GRPC_SERVICE_HDRS "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h")

    add_custom_command(
      OUTPUT "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.cc"
             "${PROJECT_BINARY_DIR}/${DIR}/${FIL_WE}.pb.h"
      COMMAND  ${PROTOBUF_PROTOC_EXECUTABLE}
      ARGS --cpp_out  ${PROJECT_BINARY_DIR}
        -I ${PROJECT_SOURCE_DIR}
        ${ABS_FIL}
      DEPENDS ${ABS_FIL}
      COMMENT "Running C++ protocol buffer compiler on ${ABS_FIL}"
      VERBATIM
    )
  endforeach()
  set_source_files_properties(${ALL_GRPC_SERVICE_SRCS} ${ALL_GRPC_SERVICE_HDRS} PROPERTIES GENERATED TRUE)
  list(APPEND ALL_LIBRARY_HDRS ${ALL_GRPC_SERVICE_HDRS})
  list(APPEND ALL_LIBRARY_SRCS ${ALL_GRPC_SERVICE_SRCS})
endif()
set(INSTALL_GENERATED_HDRS ${ALL_PROTO_HDRS} ${ALL_GRPC_SERVICE_HDRS})

# 生成cartographer库文件
add_library(${PROJECT_NAME} STATIC ${ALL_LIBRARY_HDRS} ${ALL_LIBRARY_SRCS})

configure_file(
  ${PROJECT_SOURCE_DIR}/cartographer/common/config.h.cmake
  ${PROJECT_BINARY_DIR}/cartographer/common/config.h)

google_binary(cartographer_autogenerate_ground_truth
  SRCS
    cartographer/ground_truth/autogenerate_ground_truth_main.cc
)

google_binary(cartographer_compute_relations_metrics
  SRCS
    cartographer/ground_truth/compute_relations_metrics_main.cc
)

google_binary(cartographer_pbstream
  SRCS
  cartographer/io/pbstream_main.cc
)

google_binary(cartographer_print_configuration
  SRCS
  cartographer/common/print_configuration_main.cc
)

if(${BUILD_GRPC})
  google_binary(cartographer_grpc_server
    SRCS
```

```
      cartographer/cloud/map_builder_server_main.cc
  )
  target_link_libraries(cartographer_grpc_server PUBLIC grpc++)
  target_link_libraries(cartographer_grpc_server PUBLIC async_grpc)
  if(${BUILD_PROMETHEUS})
    target_link_libraries(cartographer_grpc_server PUBLIC ${ZLIB_LIBRARIES})
    target_link_libraries(cartographer_grpc_server PUBLIC prometheus-cpp-core)
    target_link_libraries(cartographer_grpc_server PUBLIC prometheus-cpp-pull)
  endif()
endif()

target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
  "${EIGEN3_INCLUDE_DIR}")
target_link_libraries(${PROJECT_NAME} PUBLIC ${EIGEN3_LIBRARIES})

target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
  "${CERES_INCLUDE_DIRS}")
target_link_libraries(${PROJECT_NAME} PUBLIC ${CERES_LIBRARIES})

target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
  "${LUA_INCLUDE_DIR}")
target_link_libraries(${PROJECT_NAME} PUBLIC ${LUA_LIBRARIES})

target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
  "${Boost_INCLUDE_DIRS}")
target_link_libraries(${PROJECT_NAME} PUBLIC ${Boost_LIBRARIES})

if (WIN32)
  find_package(glog REQUIRED)
  set(GLOG_LIBRARY glog::glog)
else()
  set(GLOG_LIBRARY glog)
endif()

target_link_libraries(${PROJECT_NAME} PUBLIC ${GLOG_LIBRARY})
target_link_libraries(${PROJECT_NAME} PUBLIC gflags)
if(WIN32)
  # Needed to fix conflict with MSVC's error macro.
  target_compile_definitions(${PROJECT_NAME} PUBLIC -DGLOG_NO_ABBREVIATED_SEVERITIES)
endif()
if(MSVC)
  # Needed for VS 2017 5.8
  target_compile_definitions(${PROJECT_NAME} PUBLIC -D_ENABLE_EXTENDED_ALIGNED_STORAGE -D_USE_MATH_DEFINES)
endif()
```

```

if("${CAIRO_INCLUDE_DIRS}")
  target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
    "${CAIRO_INCLUDE_DIRS}")
endif()
target_link_libraries(${PROJECT_NAME} PUBLIC ${CAIRO_LIBRARIES})

target_include_directories(${PROJECT_NAME} SYSTEM PUBLIC
  ${PROTOBUF_INCLUDE_DIR})
# TODO(hrapp): This should not explicitly list pthread and use
# PROTOBUF_LIBRARIES, but that failed on first try.
target_link_libraries(${PROJECT_NAME} PUBLIC ${PROTOBUF_LIBRARY}
  absl::algorithm
  absl::base
  absl::debugging
  absl::flat_hash_map
  absl::memory
  absl::meta
  absl::numeric
  absl::str_format
  absl::strings
  absl::synchronization
  absl::time
  absl::utility
)
if (NOT WIN32)
  target_link_libraries(${PROJECT_NAME} PUBLIC pthread)
endif()
if(${BUILD_GRPC})
  target_link_libraries(${PROJECT_NAME} PUBLIC grpc++)
  target_link_libraries(${PROJECT_NAME} PUBLIC async_grpc)
endif()
if(${BUILD_PROMETHEUS})
  target_link_libraries(${PROJECT_NAME} PUBLIC ${ZLIB_LIBRARIES})
  target_link_libraries(${PROJECT_NAME} PUBLIC prometheus-cpp-core)
  target_link_libraries(${PROJECT_NAME} PUBLIC prometheus-cpp-pull)
  target_compile_definitions(${PROJECT_NAME} PUBLIC USE_PROMETHEUS=1)
endif()

set(TARGET_COMPILE_FLAGS "${TARGET_COMPILE_FLAGS} ${GOOG_CXX_FLAGS}")
set_target_properties(${PROJECT_NAME} PROPERTIES
  COMPILE_FLAGS ${TARGET_COMPILE_FLAGS})

# 不编译test_library

# set(TEST_LIB
#   cartographer_test_library
# )
```

```cmake
# add_library(${TEST_LIB} ${TEST_LIBRARY_HDRS} ${TEST_LIBRARY_SRCS})
# target_include_directories(${TEST_LIB} SYSTEM PRIVATE
#   "${GMOCK_INCLUDE_DIRS}")
# # Needed for dynamically linked GTest on Windows.
# if (WIN32)
#   target_compile_definitions(${TEST_LIB} PUBLIC -DGTEST_LINKED_AS_SHARED_LIBRARY)
# endif()
# target_link_libraries(${TEST_LIB} PUBLIC ${GMOCK_LIBRARY})
# target_link_libraries(${TEST_LIB} PUBLIC ${PROJECT_NAME})
# set_target_properties(${TEST_LIB} PROPERTIES
#   COMPILE_FLAGS ${TARGET_COMPILE_FLAGS})

# 不编译_test.cc文件

# foreach(ABS_FIL ${ALL_TESTS})
#   file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${ABS_FIL})
#   get_filename_component(DIR ${REL_FIL} DIRECTORY)
#   get_filename_component(FIL_WE ${REL_FIL} NAME_WE)
#   # Replace slashes as required for CMP0037.
#   string(REPLACE "/" "." TEST_TARGET_NAME "${DIR}/${FIL_WE}")
#   google_test("${TEST_TARGET_NAME}" ${ABS_FIL})
#   if(${BUILD_GRPC})
#     target_link_libraries("${TEST_TARGET_NAME}" PUBLIC grpc++)
#     target_link_libraries("${TEST_TARGET_NAME}" PUBLIC async_grpc)
#   endif()
#   if(${BUILD_PROMETHEUS})
#     target_link_libraries("${TEST_TARGET_NAME}" PUBLIC ${ZLIB_LIBRARIES})
#     target_link_libraries("${TEST_TARGET_NAME}" PUBLIC prometheus-cpp-core)
#     target_link_libraries("${TEST_TARGET_NAME}" PUBLIC prometheus-cpp-pull)
#   endif()
#   target_link_libraries("${TEST_TARGET_NAME}" PUBLIC ${TEST_LIB})
# endforeach()

# Add the binary directory first, so that port.h is included after it has
# been generated.
target_include_directories(${PROJECT_NAME} PUBLIC
    $<BUILD_INTERFACE:${PROJECT_BINARY_DIR}>
    $<BUILD_INTERFACE:${PROJECT_SOURCE_DIR}>
    $<INSTALL_INTERFACE:include>
)

# 安装cartographer库
install(
  TARGETS ${PROJECT_NAME}
  EXPORT CartographerExport
  ARCHIVE DESTINATION lib
  LIBRARY DESTINATION lib
  RUNTIME DESTINATION bin
)

# 安装头文件
foreach(HDR ${INSTALL_SOURCE_HDRS})
  file(RELATIVE_PATH REL_FIL ${PROJECT_SOURCE_DIR} ${HDR})
  get_filename_component(DIR ${REL_FIL} DIRECTORY)
  install(
    FILES ${HDR}
    DESTINATION include/${DIR}
  )
endforeach()

foreach(HDR ${INSTALL_GENERATED_HDRS})
  file(RELATIVE_PATH REL_FIL ${PROJECT_BINARY_DIR} ${HDR})
  get_filename_component(DIR ${REL_FIL} DIRECTORY)
  install(
    FILES ${HDR}
    DESTINATION include/${DIR}
  )
endforeach()

set(CARTOGRAPHER_CMAKE_DIR share/cartographer/cmake)
include(CMakePackageConfigHelpers)
configure_package_config_file(
  cartographer-config.cmake.in
  ${PROJECT_BINARY_DIR}/cmake/cartographer/cartographer-config.cmake
  PATH_VARS CARTOGRAPHER_CMAKE_DIR
  INSTALL_DESTINATION ${CMAKE_INSTALL_PREFIX}/share/cartographer
)

install(
  EXPORT CartographerExport
  DESTINATION share/cartographer/cmake/
  FILE CartographerTargets.cmake
)

install(
  FILES ${PROJECT_BINARY_DIR}/cmake/cartographer/cartographer-config.cmake
  DESTINATION share/cartographer/
)
```