AGVC SDK API 文档

July 1, 2026

Contents

Chapter 1

AGVC SDK

概述

AGVC SDK 提供了一套完整的接口来控制和管理AGV 系统，通过 AGVC_API (主入口) 建立通讯可以实现AGV 的地图管理、路径规划、运动控制、充电、系统状态监控等功能

目录

快速开始

基本使用流程

```
// 1. agv sdk 初始化
auto rpc = std::make_shared<agvc_interface::RpcClient>();

// 2. agv 通讯设置
// 2.1 设置通讯超时时间
rpc->setRequestTimeout(100);
// 2.2 连接 agv
rpc->connect("192.168.192.100", 30104);
// 2.3 登录
rpc->login("aa", "bb");

// 3. agv 功能接口使用
// 3.1 查询 agv 信息
rpc->getAgvDetails();
// 3.2 获取激光点云
rpc->getLaserPointCloud();
```

SDK 架构

层次结构

```
AGVC_API (主入口)
  Map(地图)
  Station(站点)
  Path(路径)
  Navigation(导航)
```

Charging(充电)
AgvInfo(Agv 信息)
System(系统)

使用示例

获取机器人详细信息

```
// 建立通讯
auto rpc = std::make_shared<agvc_interface::RpcClient>();
rpc->setRequestTimeout(100);
rpc->connect("192.168.192.100", 30104);
rpc->login("aa", "bb");

// 获取 agv 详细信息
auto agv_details = rpc->getAgvDetails();
```

创建地图

```
// 抢占控制权
rpc->setPriority("aa", "");

// 切换建图模式
agvc_interface::Header header;
header.id = "123sdaw-asdafas-zff";
agvc_interface::RunningMode mode = agvc_interface::RunningMode::MAPPING;
auto ret = rpc->changeRunningMode(mode, header);

// 查询异步接口 (changeRunningMode) 执行状态
agvc_interface::AsyncInterfaceResultStatus async_rusult = rpc->getAsyncInterfaceResultStatus();
agvc_interface::ChangeRunningModeStatus change_running_mode_status = async_rusult.change_running_mode_status.status;

// 如果异步接口运行状态为 Running，则重复调用，直到异步接口运行完成
while (change_running_mode_status == agvc_interface::ChangeRunningModeStatus::RUNNING)
{
    change_running_mode_status = rpc->getAsyncInterfaceResultStatus().change_running_mode_status.status;
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}

// 异步接口运行状态非 running，接口运行结束，重新调用接口查询执行接口
ret = rpc->changeRunningMode(mode, header);

// 判断异步接口运行结果
if(ret == 10100000){
    std::cout <<" 切换建图模式成功";
}else{
    std::cout <<" 切换建图模式失败，错误码: "<< ret;
}
```

保存地图

```
// 抢占控制权
rpc->setPriority("aa", "");

// 设置地图 header
agvc_interface::Header map_header;
map_header.id     = "test_save_map";
map_header.map_id = map_id;
map_header.name   = "test1";
map_header.stamp  = 1;

auto ret = rpc->saveMap(map_header);

// 查询异步接口 (savemap) 执行状态
agvc_interface::AsyncInterfaceResultStatus async_rusult = rpc->getAsyncInterfaceResultStatus();
auto save_map_status = ret.save_map_status.status;

// 如果异步接口运行状态为 Running，则重复调用，直到异步接口运行完成
while (save_map_status == agvc_interface::ChangeRunningModeStatus::RUNNING)
{
    save_map_status = rpc->getAsyncInterfaceResultStatus().save_map_status.status;
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}

// 异步接口运行状态非 running，接口运行结束，重新调用接口查询执行接口
ret = rpc->saveMap(map_header);

// 判断异步接口运行结果
if(ret == 10100000){
    std::cout <<" 保存地图成功";
```

```
}else{
    std::cout <<" 保存地图失败，错误码："<< ret;
}
```

命名空间

所有接口都定义在以下命名空间中：

```
namespace agvc_interface {
    // 所有接口定义
}
```

错误处理

大多数接口方法返回整数错误码，其中 10100000 表示成功，其他值表示错误码，可以通过错误码查询接口查询含义。某些接口可能抛出 AuboException 异常。

弃用列表

已弃用接口和字段请参考 deprecated_zh。

支持的编程语言

- C++
- Python
- JSON-RPC

相关文件

- agvc_interface.h - 接口定义
- type.h - 类型定义

Chapter 2

弃用列表

弃用列表

本页汇总当前 SDK 中已弃用的 C++ 接口和字段。请优先使用“替代接口/字段”。

已弃用项	自版本	替代接口/字段	说明
agvc_interface::Agvc↩ Interface::checkPower↩ AndAutoCharge()	0.7.0	sendAutoCharging↩ Command(const AutoChargingCommand&)	检测当前电量，电量低将自动上桩充电。
agvc_interface::Agvc↩ Interface::setLeave↩ BoardTargetStation()	0.7.0	sendAutoCharging↩ Command(const AutoChargingCommand&)	设置自动下桩时的目标站点。
agvc_interface::Agvc↩ Interface::getLeave↩ BoardTargetStation()	0.7.0	getNavInfo()	获取自动下桩时的目标站点。
agvc_interface::Agvc↩ Interface::forcedAgvTo↩ ChargingBoard()	0.7.0	sendAutoCharging↩ Command(const AutoChargingCommand&)	强制上桩充电。
agvc_interface::Agvc↩ Interface::forcedAgv↩ LeaveChargingBoard()	0.7.0	sendAutoCharging↩ Command(const AutoChargingCommand&)	强制下桩。
agvc_interface::Nav↩ Info::type	0.7.0	nav_type	导航类型字段已由 nav_type 替代。

Chapter 3

专题索引

3.1 专题

这里是所有专题及其简介:

Chapter 4

目录结构

4.1 目录

Chapter 5

命名空间索引

5.1 命名空间列表

这里列出了所有命名空间定义，附带简要说明:

Chapter 6

继承关系索引

6.1 类继承关系

此继承关系列表按字典顺序粗略的排序:

Chapter 7

类索引

7.1 类列表

这里列出了所有类、结构、联合以及接口定义等，并附带简要说明:

Chapter 8

文件索引

8.1 文件列表

这里列出了所有文件，并附带简要说明:

Chapter 9

专题文档

9.1 Map (地图模块)

函数

- std::vector< Header > agvc_interface::AgvcInterface::getMapList ()

  获取*AGV* 所有地图的信息头。所有地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。
- Header agvc_interface::AgvcInterface::getCurrentMapHeader ()

  查询当前*AGV* 地图的信息头。当前地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。
- OccupancyGridMap agvc_interface::AgvcInterface::getGridMapFromAgv (const Header &map_↩ header)

  获取指定栅格地图信息。指定地图的信息头 *(*本次异步命令 *id*, 名称, 时间, 地图 *id)*。*map_id* 为*"current*↩ *_map"* 时，可特指为当前地图。指定栅格地图信息。
- Base64PngMap agvc_interface::AgvcInterface::getBase64PngMapFromAgv (const Header &map↩ _header)
- Base64PngMap agvc_interface::AgvcInterface::previewPngMapFromAgv (const Header &map_↩ header, const int &image_width_px=0, const int &image_height_px=0)
- int agvc_interface::AgvcInterface::sendGridMapToAgv (const OccupancyGridMap &map)
- int agvc_interface::AgvcInterface::sendBase64PngMapToAgv (const Base64PngMap &map)
- int agvc_interface::AgvcInterface::saveMap (const Header &map_header)
- int agvc_interface::AgvcInterface::switchMap (const Header &map_header)
- int agvc_interface::AgvcInterface::deleteMap (const Header &map_header)

  删除一张指定地图。指定地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除地图成功; *else*: 删除地图失败。
- int agvc_interface::AgvcInterface::deleteMaps (const std::vector< Header > &map_headers)

  删除多张指定地图。多个指定地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除地图成功; *else*: 删除地图失败。
- std::vector< MapVirtualArea > agvc_interface::AgvcInterface::getAllMapVirtualArea ()

  查询当前地图的虚拟区域。当前地图的虚拟区域信息。
- std::vector< MapVirtualArea > agvc_interface::AgvcInterface::getAllMapVirtualAreaOfTargetMap (const Header &map_header)

  查询目标地图的虚拟区域。指定地图 *header(*命令 *id*, 名称, 时间, 地图 *id)*。指定地图的所有虚拟区域。
- int agvc_interface::AgvcInterface::addMapVirtualArea (const MapVirtualArea &map_virtual_↩ area)

  添加或修改一个虚拟区域。待添加或修改的虚拟区域信息。*10100000*: 添加或修改虚拟区域成功; *else*: 添加或修改虚拟区域失败。
- int agvc_interface::AgvcInterface::addMapVirtualAreas (const std::vector< MapVirtualArea > &map↩ _virtual_areas)

  添加或修改多个虚拟区域。待添加或修改的虚拟区域信息。*10100000*: 添加或修改虚拟区域成功; *else*: 添加或修改虚拟区域失败。
- int agvc_interface::AgvcInterface::deleteMapVirtualArea (const Header &virtual_area_header)

删除指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*，虚拟区域名称，时间，所在地图 *id)*。*10100000*：删除虚拟区域成功；*else*：删除虚拟区域失败。

- int agvc_interface::AgvcInterface::deleteMapVirtualAreas (const std::vector< Header > &virtual↩ _areas_header)

  删除多个指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*，虚拟区域名称，时间，所在地图 *id)*。*10100000*：删除虚拟区域成功；*else*：删除虚拟区域失败。

- MapAllInfo agvc_interface::AgvcInterface::getGridMapAllInfo (const Header &map_header)
- MapAllInfo agvc_interface::AgvcInterface::getPngMapAllInfo (const Header &map_header)
- int agvc_interface::AgvcInterface::setGridMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int agvc_interface::AgvcInterface::setPngMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int agvc_interface::AgvcInterface::deleteMapAllInfo (const Header &map_header)

  删除指定地图的全部信息，包括地图数据、站点、路径、虚拟区信息。指定地图信息头 *(*命令 *id*，地图名称，时间，地图 *id)*。*10100000*：删除成功；*else*：删除失败。

9.1.1 详细描述

AGVC 自定义地图的创建、切换、删除、数据读写等核心功能

9.1.2 函数说明

addMapVirtualArea()

```
int agvc_interface::AgvcInterface::addMapVirtualArea (
            const MapVirtualArea & map_virtual_area)
```

添加或修改一个虚拟区域。待添加或修改的虚拟区域信息。10100000：添加或修改虚拟区域成功；else：添加或修改虚拟区域失败。

addMapVirtualAreas()

```
int agvc_interface::AgvcInterface::addMapVirtualAreas (
            const std::vector< MapVirtualArea > & map_virtual_areas)
```

添加或修改多个虚拟区域。待添加或修改的虚拟区域信息。10100000：添加或修改虚拟区域成功；else：添加或修改虚拟区域失败。

deleteMap()

```
int agvc_interface::AgvcInterface::deleteMap (
            const Header & map_header)
```

删除一张指定地图。指定地图的信息头 (命令 id，名称，时间，地图 id)。10100000：删除地图成功；else：删除地图失败。

deleteMapAllInfo()

```
int agvc_interface::AgvcInterface::deleteMapAllInfo (
            const Header & map_header)
```

删除指定地图的全部信息，包括地图数据、站点、路径、虚拟区信息。指定地图信息头 (命令 id，地图名称，时间，地图 id)。10100000：删除成功；else：删除失败。

deleteMaps()

```
int agvc_interface::AgvcInterface::deleteMaps (
            const std::vector< Header > & map_headers)
```

删除多张指定地图。多个指定地图的信息头 (命令 id，名称，时间，地图 id)。10100000：删除地图成功；else：删除地图失败。

deleteMapVirtualArea()

```
int agvc_interface::AgvcInterface::deleteMapVirtualArea (
        const Header & virtual_area_header)
```

删除指定虚拟区域。指定虚拟区域信息头 (命令 id，虚拟区域名称，时间，所在地图 id)。10100000：删除虚拟区域成功；else：删除虚拟区域失败。

deleteMapVirtualAreas()

```
int agvc_interface::AgvcInterface::deleteMapVirtualAreas (
        const std::vector< Header > & virtual_areas_header)
```

删除多个指定虚拟区域。指定虚拟区域信息头 (命令 id, 虚拟区域名称, 时间, 所在地图 id)。10100000：删除虚拟区域成功；else：删除虚拟区域失败。

getAllMapVirtualArea()

```
std::vector< MapVirtualArea > agvc_interface::AgvcInterface::getAllMapVirtualArea ()
```

查询当前地图的虚拟区域。当前地图的虚拟区域信息。

getAllMapVirtualAreaOfTargetMap()

```
std::vector< MapVirtualArea > agvc_interface::AgvcInterface::getAllMapVirtualAreaOfTargetMap (
        const Header & map_header)
```

查询目标地图的虚拟区域。指定地图 header(命令 id，名称，时间，地图 id)。指定地图的所有虚拟区域。

getBase64PngMapFromAgv()

```
Base64PngMap agvc_interface::AgvcInterface::getBase64PngMapFromAgv (
        const Header & map_header)
```

注意

1. 异步接口。获取Base64 编码的PNG 地图信息。指定地图的信息头 (本次异步命令 id，名称，时间，地图 id)。map_id 为"current_map" 时，可特指为当前地图。Base64 编码的PNG 地图信息。

getCurrentMapHeader()

```
Header agvc_interface::AgvcInterface::getCurrentMapHeader ()
```

查询当前AGV 地图的信息头。当前地图的信息头 (命令 id，名称，时间，地图 id)。

getGridMapAllInfo()

```
MapAllInfo agvc_interface::AgvcInterface::getGridMapAllInfo (
        const Header & map_header)
```

注意

1. 异步接口。获取指定地图的栅格地图数据、路径、站点、虚拟区信息。指定地图信息头 (本次异步命令 id，地图名称，时间，地图 id)。map_id 为"current_map" 时，可特指为当前地图。栅格地图上的全部信息。

getGridMapFromAgv()

```
OccupancyGridMap agvc_interface::AgvcInterface::getGridMapFromAgv (
        const Header & map_header)
```

获取指定栅格地图信息。指定地图的信息头 (本次异步命令 id，名称，时间，地图 id)。map_id 为"current_map" 时，可特指为当前地图。指定栅格地图信息。

getMapList()

```
std::vector< Header > agvc_interface::AgvcInterface::getMapList ()
```

获取AGV 所有地图的信息头。所有地图的信息头 (命令 id，名称，时间，地图 id)。

getPngMapAllInfo()

```
MapAllInfo agvc_interface::AgvcInterface::getPngMapAllInfo (
            const Header & map_header)
```

注意

1. 异步接口。获取指定地图的Base64 地图数据、路径、站点、虚拟区信息。指定地图信息头 (本次异步命令 id，地图名称，时间，地图 id)。map_id 为"current_map" 时，可特指为当前地图。Base64 地图上的全部信息。

previewPngMapFromAgv()

```
Base64PngMap agvc_interface::AgvcInterface::previewPngMapFromAgv (
            const Header & map_header,
            const int & image_width_px = 0,
            const int & image_height_px = 0)
```

注意

1. 异步接口。获取预览图，可指定预览图的长和宽。指定地图的信息头 (本次异步命令 id，名称，时间，地图 id)。地图宽度 (像素)，默认值为 0 时返回缩略图数据为空。地图高度 (像素)，默认值为 0 时返回缩略图数据为空。map_id 为"current_map" 时，可特指为当前地图。Base64 编码的PNG 缩略图。

saveMap()

```
int agvc_interface::AgvcInterface::saveMap (
            const Header & map_header)
```

注意

1. 异步接口；2.控制权限制。建图完成后调用该接口保存地图。地图信息头 (本次异步命令 id，名称，时间，地图 id)。10100000：保存地图成功；10100201：异步接口运行中；10120202：无控制权；else：保存地图失败。

sendBase64PngMapToAgv()

```
int agvc_interface::AgvcInterface::sendBase64PngMapToAgv (
            const Base64PngMap & map)
```

注意

1. 异步接口。将Base64 编码的PNG 地图发送给AGV。Base64 编码的PNG 地图。10100000：发送地图成功；10100201：发送地图中；else：发送地图失败。

sendGridMapToAgv()

```
int agvc_interface::AgvcInterface::sendGridMapToAgv (
            const OccupancyGridMap & map)
```

注意

1. 异步接口。将栅格地图信息发送给AGV。栅格地图信息。10100000：发送地图成功；10100201：发送地图中；else：发送地图失败。

setGridMapAllInfo()

```
int agvc_interface::AgvcInterface::setGridMapAllInfo (
            const MapAllInfo & map_all_info,
            const Header & command_header = { "99999", "99999", 1, "99999" })
```

注意

1. 异步接口；2.控制权限制。设置栅格地图、路径、站点、虚拟区信息。栅格地图上的全部信息。id 字段代表下发本次命令的 id；其他字段无特殊含义。请保证数据下发的可行性，AGV 不会对数据进行校验。10100000：上传全部地图信息成功；10100201：异步接口运行中；10120202：无控制权；else：上传全部地图信息失败。

setPngMapAllInfo()

```
int agvc_interface::AgvcInterface::setPngMapAllInfo (
            const MapAllInfo & map_all_info,
            const Header & command_header = { "99999", "99999", 1, "99999" })
```

注意

1. 异步接口；2.控制权限制。设置Base64 格式地图、路径、站点、虚拟区信息。Base64 格式地图上的全部信息。id 字段代表下发本次命令的 id；其他字段无特殊含义。请保证数据下发的可行性，AGV 不会对数据进行校验。10100000：上传全部地图信息成功；10100201：异步接口运行中；10120202：无控制权；else：上传全部地图信息失败。

switchMap()

```
int agvc_interface::AgvcInterface::switchMap (
            const Header & map_header)
```

注意

1. 异步接口；2.控制权限制。切换指定地图。指定地图的信息头 (本次异步命令 id，名称，时间，地图 id)。10100000：切换地图成功；10100201：异步接口运行中；10120202：无控制权；else：切换地图失败。

9.2 Station (站点模块)

函数

- std::vector< StationMark > agvc_interface::AgvcInterface::getAllStations ()

  查询当前地图的所有站点信息。当前地图的所有站点信息。
- std::vector< StationMark > agvc_interface::AgvcInterface::getAllStationsOfTargetMap (const Header &map_header)

  查询指定地图的所有站点信息。指定地图 *header(*命令 *id*，名称，时间，地图 *id)*。指定地图的所有站点信息。
- int agvc_interface::AgvcInterface::addStation (const StationMark &station)

  添加或修改一个站点。待添加或修改的站点信息。*10100000*：添加或修改站点成功；*else*：添加或修改站点失败。
- int agvc_interface::AgvcInterface::addStations (const std::vector< StationMark > &stations)

  添加或修改多个站点。待添加或修改的站点信息。*10100000*：添加或修改站点成功；*else*：添加或修改站点失败。
- int agvc_interface::AgvcInterface::addCPStationUseChargingBoard (const Header &station_header, const double &dis_station_board=1.5)

  通过自动识别充电桩位姿来添加充电站点。待添加或修改的充电站点信息 *(*站点 *id*，名称，时间，地图 *id)*。充电站点与充电桩之间的距离 *(*充电站点位于*AGV* 中心*)*，单位：*m*，范围*[1.0∼2.0]*。*10100000*：添加充电站点成功；*else*：添加充电站点失败。
- int agvc_interface::AgvcInterface::deleteStation (const Header &station_header)

删除指定站点，会删除与该站点相关的路径。待删除站点信息 *(*站点 *id*，名称，时间，地图 *id)*。*10100000*: 删除站点成功；*else*：删除站点失败。

- int agvc_interface::AgvcInterface::deleteStations (const std::vector< Header > &stations_header)

  删除多个站点，会删除与站点相关的路径。待删除站点信息 *(*站点 *id*，名称，时间，地图 *id)*。*10100000*: 删除站点成功；*else*：删除站点失败。

9.2.1 详细描述

AGVC 地图站点的增删改查、充电站点自动识别等功能

9.2.2 函数说明

addCPStationUseChargingBoard()

```
int agvc_interface::AgvcInterface::addCPStationUseChargingBoard (
        const Header & station_header,
        const double & dis_station_board = 1.5)
```

通过自动识别充电桩位姿来添加充电站点。待添加或修改的充电站点信息 (站点 id，名称，时间，地图 id)。充电站点与充电桩之间的距离 (充电站点位于AGV 中心)，单位：m，范围[1.0~2.0]。10100000: 添加充电站点成功；else：添加充电站点失败。

addStation()

```
int agvc_interface::AgvcInterface::addStation (
        const StationMark & station)
```

添加或修改一个站点。待添加或修改的站点信息。10100000：添加或修改站点成功；else：添加或修改站点失败。

addStations()

```
int agvc_interface::AgvcInterface::addStations (
        const std::vector< StationMark > & stations)
```

添加或修改多个站点。待添加或修改的站点信息。10100000：添加或修改站点成功；else：添加或修改站点失败。

deleteStation()

```
int agvc_interface::AgvcInterface::deleteStation (
        const Header & station_header)
```

删除指定站点, 会删除与该站点相关的路径。待删除站点信息 (站点 id, 名称, 时间, 地图 id)。10100000: 删除站点成功；else：删除站点失败。

deleteStations()

```
int agvc_interface::AgvcInterface::deleteStations (
        const std::vector< Header > & stations_header)
```

删除多个站点，会删除与站点相关的路径。待删除站点信息 (站点 id，名称，时间，地图 id)。10100000: 删除站点成功；else：删除站点失败。

getAllStations()

```
std::vector< StationMark > agvc_interface::AgvcInterface::getAllStations ()
```

查询当前地图的所有站点信息。当前地图的所有站点信息。

getAllStationsOfTargetMap()

```
std::vector< StationMark > agvc_interface::AgvcInterface::getAllStationsOfTargetMap (
        const Header & map_header)
```

查询指定地图的所有站点信息。指定地图 header(命令 id，名称，时间，地图 id)。指定地图的所有站点信息。

9.3 Path (路径模块)

函数

- std::vector< PathStation > agvc_interface::AgvcInterface::getAllPaths ()

  查询当前地图的所有路径信息。当前地图的所有路径信息。
- std::vector< PathStation > agvc_interface::AgvcInterface::getAllPathsOfTargetMap (const Header &map_header)

  查询指定地图的所有路径信息。指定地图 *header(*命令 *id*，名称，时间，地图 *id)*。指定地图的所有路径信息。
- PathStation agvc_interface::AgvcInterface::getCurrentPath ()

  查询*AGV* 当前正在跟踪的路径信息。当前正在跟踪的路径信息。
- int agvc_interface::AgvcInterface::generatePath (const PathStation &path_station)

  生成或修改一条路径。待生成或修改的路径信息。*10100000*：修改或生成路径成功；*else*：修改或生成路径失败。
- int agvc_interface::AgvcInterface::generatePaths (const std::vector< PathStation > &paths_station)

  生成或修改多条路径。待生成或修改的路径信息。*10100000*：修改或生成路径成功；*else*：修改或生成路径失败。
- int agvc_interface::AgvcInterface::deletePath (const Header &path_header)

  删除一条路径。待删除路径信息 *(*路径 *id*，名称，时间，地图 *id)*。*10100000*：删除路径成功；*else*：删除路径失败。
- int agvc_interface::AgvcInterface::deletePaths (const std::vector< Header > &paths_header)

  删除多条路径。待删除路径信息 *(*路径 *id*，名称，时间，地图 *id)*。*10100000*：删除路径成功；*else*：删除路径失败。

9.3.1 详细描述

AGVC 地图路径的生成、修改、删除、查询等功能

9.3.2 函数说明

deletePath()

```
int agvc_interface::AgvcInterface::deletePath (
            const Header & path_header)
```

删除一条路径。待删除路径信息 (路径 id，名称，时间，地图 id)。10100000：删除路径成功；else：删除路径失败。

deletePaths()

```
int agvc_interface::AgvcInterface::deletePaths (
            const std::vector< Header > & paths_header)
```

删除多条路径。待删除路径信息 (路径 id，名称，时间，地图 id)。10100000：删除路径成功；else：删除路径失败。

generatePath()

```
int agvc_interface::AgvcInterface::generatePath (
            const PathStation & path_station)
```

生成或修改一条路径。待生成或修改的路径信息。10100000：修改或生成路径成功；else：修改或生成路径失败。

generatePaths()

```
int agvc_interface::AgvcInterface::generatePaths (
            const std::vector< PathStation > & paths_station)
```

生成或修改多条路径。待生成或修改的路径信息。10100000：修改或生成路径成功；else：修改或生成路径失败。

getAllPaths()

```
std::vector< PathStation > agvc_interface::AgvcInterface::getAllPaths ()
```

查询当前地图的所有路径信息。当前地图的所有路径信息。

getAllPathsOfTargetMap()

```
std::vector< PathStation > agvc_interface::AgvcInterface::getAllPathsOfTargetMap (
        const Header & map_header)
```

查询指定地图的所有路径信息。指定地图 header(命令 id，名称，时间，地图 id)。指定地图的所有路径信息。

getCurrentPath()

```
PathStation agvc_interface::AgvcInterface::getCurrentPath ()
```

查询AGV 当前正在跟踪的路径信息。当前正在跟踪的路径信息。

9.4 Navigation (导航模块)

函数

- int agvc_interface::AgvcInterface::changeRunningMode (const RunningMode &running_mode, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int agvc_interface::AgvcInterface::setControlSpeed (const Speed &speed)
- int agvc_interface::AgvcInterface::setNavGoal (const NavGoalType &target)
- int agvc_interface::AgvcInterface::pauseAgvSpeed ()
- int agvc_interface::AgvcInterface::resumeAgvSpeed ()
- int agvc_interface::AgvcInterface::cancelNavigation ()
- int agvc_interface::AgvcInterface::relocation (const Pose2d &init_pose, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

9.4.1 详细描述

AGVC 运动控制、导航任务下发、重定位、模式切换等功能

9.4.2 函数说明

cancelNavigation()

```
int agvc_interface::AgvcInterface::cancelNavigation ()
```

注意

控制权限制。取消导航任务。10100000：取消导航任务成功；10120202：没有控制权；else：取消导航任务失败。

changeRunningMode()

```
int agvc_interface::AgvcInterface::changeRunningMode (
        const RunningMode & running_mode,
        const Header & command_header = { "99999", "99999", 1, "99999" })
```

注意

1. 异步接口；2.控制权限制。切换AGV 模式。打算切换的模式，如导航模式、建图模式。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：切换模式成功；10100201：异步接口运行中；10120202：无控制权；else：切换模式失败。

pauseAgvSpeed()

```
int agvc_interface::AgvcInterface::pauseAgvSpeed ()
```

注意

控制权限制。暂停AGV 速度。10100000：暂停成功；10120202：没有控制权；else：暂停失败。

relocation()

```
int agvc_interface::AgvcInterface::relocation (
            const Pose2d & init_pose,
            const Header & command_header = { "99999", "99999", 1, "99999" })
```

注意

1. 异步接口；2.控制权限制。AGV 重定位。AGV 当前参考位姿。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：重定位成功；10100201：异步接口运行中；10120202：无控制权；else：重定位失败。

resumeAgvSpeed()

```
int agvc_interface::AgvcInterface::resumeAgvSpeed ()
```

注意

控制权限制。暂停后恢复AGV 速度。10100000：恢复成功；10120202：没有控制权；else：恢复失败。

setControlSpeed()

```
int agvc_interface::AgvcInterface::setControlSpeed (
            const Speed & speed)
```

注意

控制权限制。控制AGV 运动。速度信息。10100000：向AGV 下发速度成功；10120202：没有控制权；else：向AGV 下发速度失败。

setNavGoal()

```
int agvc_interface::AgvcInterface::setNavGoal (
            const NavGoalType & target)
```

注意

控制权限制。向AGV 发送导航任务。目标位置信息与导航参数信息。10100000：设置导航任务成功；10120202：没有控制权；else：设置导航任务失败。

9.5 Charging (充电模块)

函数

- int agvc_interface::AgvcInterface::checkPowerAndAutoCharge (const Header &check_power_header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &nav_board_charge_station_id=”99999”)
- int agvc_interface::AgvcInterface::setLeaveBoardTargetStation (const Header &leave_board_target↩ _station_header)

  设置自动下桩时的目标站点。自动下桩的站点 *header* 信息。*10100000*：设置自动下桩站点成功；*else*：设置自动下桩站点失败。
- Header agvc_interface::AgvcInterface::getLeaveBoardTargetStation ()

  获取自动下桩时的目标站点。自动下桩时的目标站点*Header* 信息。

- int agvc_interface::AgvcInterface::forcedAgvToChargingBoard (const Header &forced_charge_↩ header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &nav_board_charge_station_↩ id=”99999”)
- int agvc_interface::AgvcInterface::forcedAgvLeaveChargingBoard (const Header &forced_leave↩ _header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &leave_board_target_station↩ _id=”99999”)
- int agvc_interface::AgvcInterface::sendAutoChargingCommand (const AutoChargingCommand &auto↩ _charging_command)

9.5.1 详细描述

AGVC 自动充电、强制上下桩、电量检测等充电相关功能

9.5.2 函数说明

checkPowerAndAutoCharge()

```
int agvc_interface::AgvcInterface::checkPowerAndAutoCharge (
          const Header & check_power_header = { "99999", "99999", 99999, "99999" },
          const std::string & nav_board_charge_station_id = "99999")
```

注意

控制权限制。检测当前电量，电量低将自动上桩充电。充电完成后，根据配置文件中的参数 auto_leave_board，决定是否自动下桩并行驶到指定站点或等待区。在配置文件中设置电量低阈值[low_battery_threshold]或电量高阈值[high_battery_threshold]。本次命令的 id，本次命令的 name，当前时间，地图 id。行驶到指定的充电站点进行上桩充电；如果不设置该参数，行驶到最近的充电站点进行上桩充电。10100000：接口调用成功，AGV 开始执行充电命令；10120202：无控制权；else：接口调用失败。

forcedAgvLeaveChargingBoard()

```
int agvc_interface::AgvcInterface::forcedAgvLeaveChargingBoard (
          const Header & forced_leave_header = { "99999", "99999", 99999, "99999" },
          const std::string & leave_board_target_station_id = "99999")
```

注意

控制权限制。强制下桩。本次命令的 id，本次命令的 name，当前时间，地图 id。下桩前往的站点；如果不设置该参数，则下桩到充电站点。10100000：接口调用成功，AGV 开始下桩；10120202：无控制权；else：接口调用失败。

forcedAgvToChargingBoard()

```
int agvc_interface::AgvcInterface::forcedAgvToChargingBoard (
          const Header & forced_charge_header = { "99999", "99999", 99999, "99999" },
          const std::string & nav_board_charge_station_id = "99999")
```

注意

控制权限制。强制上桩充电。本次命令的 id，本次命令的 name，当前时间，地图 id。行驶到指定的充电站点进行上桩充电；如果不设置该参数，行驶到最近的充电站点进行上桩充电。10100000：接口调用成功，AGV 开始上桩充电；10120202：无控制权；else：接口调用失败。

getLeaveBoardTargetStation()

```
Header agvc_interface::AgvcInterface::getLeaveBoardTargetStation ()
```

获取自动下桩时的目标站点。自动下桩时的目标站点Header 信息。

sendAutoChargingCommand()

```
int agvc_interface::AgvcInterface::sendAutoChargingCommand (
            const AutoChargingCommand & auto_charging_command)
```

注意

控制权限制。下发自动充电命令。自动充电命令。10100000：接口调用成功，AGV 开始执行充电命令；10120202：无控制权；else：接口调用失败。

setLeaveBoardTargetStation()

```
int agvc_interface::AgvcInterface::setLeaveBoardTargetStation (
            const Header & leave_board_target_station_header)
```

设置自动下桩时的目标站点。自动下桩的站点 header 信息。10100000：设置自动下桩站点成功；else：设置自动下桩站点失败。

9.6 AgvInfo (AGV 信息模块)

函数

- AgvDetails agvc_interface::AgvcInterface::getAgvDetails ()
- RunningInfo agvc_interface::AgvcInterface::getRunningInfo ()
- Pose2d agvc_interface::AgvcInterface::getAgvCurrentPose ()
- std::vector< Point2d > agvc_interface::AgvcInterface::getLaserPointCloud ()
- NavInfo agvc_interface::AgvcInterface::getNavInfo ()
- std::string agvc_interface::AgvcInterface::getAgvLogMessage ()

  获取*AGV* 运行中的日志信息。*AGV* 运行中的日志信息。
- ScriptRuntimeState agvc_interface::AgvcInterface::getScriptStatus ()

  获取脚本运行状态。脚本运行状态。
- std::string agvc_interface::AgvcInterface::errorCodeDecoder (const int &error_code)

  错误码查询。错误码。错误码含义；为空表示错误码未知。
- std::vector< Header > agvc_interface::AgvcInterface::getCurrentErrorCodes ()

  获取当前的错误码。错误码集合。
- int agvc_interface::AgvcInterface::soundLightPrompt ()

  寻找*AGV*，自动播放语音并特殊灯光闪烁。*10100000*：寻找指令下发成功；*else*：下发指令失败。
- bool agvc_interface::AgvcInterface::isObstacleAhead (const double &detect_distance=1.0)

  AGV 行进方向是否有障碍物。检测距离，默认为 *1.0m*。*true*：行进方向存在障碍物；*false*：行进方向无障碍物。
- StationMark agvc_interface::AgvcInterface::getNearestStation (const double &detect_distance=1.↩ 0)

  获取指定检测距离范围内的最近站点信息。检测距离，默认为 *1.0m*。最近站点信息，*id* 为*NONE* 代表检测距离内无站点。

9.6.1 详细描述

AGVC 状态、位姿、日志、错误码等信息查询功能

9.6.2 函数说明

errorCodeDecoder()

```
std::string agvc_interface::AgvcInterface::errorCodeDecoder (
            const int & error_code)
```

错误码查询。错误码。错误码含义；为空表示错误码未知。

getAgvCurrentPose()

```
Pose2d agvc_interface::AgvcInterface::getAgvCurrentPose ()
```

注意

RTDE 推送。查询AGV 当前位姿。当前位姿。

getAgvDetails()

```
AgvDetails agvc_interface::AgvcInterface::getAgvDetails ()
```

注意

RTDE 推送。查询当前AGV 信息。AGV 详细信息。

getAgvLogMessage()

```
std::string agvc_interface::AgvcInterface::getAgvLogMessage ()
```

获取AGV 运行中的日志信息。AGV 运行中的日志信息。

getCurrentErrorCodes()

```
std::vector< Header > agvc_interface::AgvcInterface::getCurrentErrorCodes ()
```

获取当前的错误码。错误码集合。

getLaserPointCloud()

```
std::vector< Point2d > agvc_interface::AgvcInterface::getLaserPointCloud ()
```

注意

RTDE 推送。查询当前激光点云数据。二维激光点坐标。

getNavInfo()

```
NavInfo agvc_interface::AgvcInterface::getNavInfo ()
```

注意

RTDE 推送。获取当前导航信息。当前导航信息。

getNearestStation()

```
StationMark agvc_interface::AgvcInterface::getNearestStation (
            const double & detect_distance = 1.0)
```

获取指定检测距离范围内的最近站点信息。检测距离，默认为 1.0m。最近站点信息，id 为NONE 代表检测距离内无站点。

getRunningInfo()

```
RunningInfo agvc_interface::AgvcInterface::getRunningInfo ()
```

注意

RTDE 推送。查询当前AGV 运行信息。AGV 运行信息。

getScriptStatus()

```
ScriptRuntimeState agvc_interface::AgvcInterface::getScriptStatus ()
```

获取脚本运行状态。脚本运行状态。

isObstacleAhead()

```
bool agvc_interface::AgvcInterface::isObstacleAhead (
        const double & detect_distance = 1.0)
```

AGV 行进方向是否有障碍物。检测距离，默认为 1.0m。true：行进方向存在障碍物；false：行进方向无障碍物。

soundLightPrompt()

```
int agvc_interface::AgvcInterface::soundLightPrompt ()
```

寻找AGV，自动播放语音并特殊灯光闪烁。10100000：寻找指令下发成功；else：下发指令失败。

9.7 System (系统模块)

函数

- int agvc_interface::AgvcInterface::setSystemClock (const std::string &stamp)

  设置*AGV* 系统时间。时间戳，格式为*YYYY-MM-DDTHH:mm:ss.ssZ*，例如“*2017-04-15T11:40:03.12Z*”。
- AsyncInterfaceResultStatus agvc_interface::AgvcInterface::getAsyncInterfaceResultStatus ()

  查询异步接口的运行情况，不代表异步接口本身的运行结果。异步接口：*saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv*。异步接口：*getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv*。异步接口：*getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv*。异步接口：*autoAlignRailway*。客户端与*AGV* 断联后，可调用该接口获取异步接口运行情况。返回值的每个字段中均有 *header*，其中 *header.id* 表示下发异步任务时的 *id* 号。*NONE*：异步接口未运行；*RUNNING*：异步接口运行中；*FINISH*：异步接口运行结束。
- std::vector< std::string > agvc_interface::AgvcInterface::getWifiList ()
- int agvc_interface::AgvcInterface::connectWifi (const std::string &ssid, const std::string &password)

  连接到指定*WiFi*，自动切换到*WiFi* 模式。必须在有线连接时调用，且执行时间较长。*WiFi* 名称。*WiFi* 密码。*10100000*：连接*WiFi* 成功；*else*：连接*WiFi* 失败。
- int agvc_interface::AgvcInterface::enableHotspot (const std::string &password)

  开启热点，自动切换到热点模式。必须在有线连接时调用，且执行时间较长。热点密码；如果为空使用*”administrator”* 为默认密码，*SSID* 默认为*SN* 码。*10100000*：启用热点成功；*else*：启用热点失败。
- std::vector< IpAddressInfo > agvc_interface::AgvcInterface::getIpAddressList ()

  获取本机所有*IP* 地址。本机所有*IP* 地址。
- std::string agvc_interface::AgvcInterface::getAgvControllerParametersFile ()

  获取*AGV* 控制器的参数文件。当前*AGV* 控制器中的参数信息。
- int agvc_interface::AgvcInterface::setAgvControllerParametersFile (const std::string &agv_parameters)
- int agvc_interface::AgvcInterface::refreshAgvControllerParametersFile ()
- int agvc_interface::AgvcInterface::resetAgvControllerParametersFile ()
- int agvc_interface::AgvcInterface::updateSoftware (const std::string &upgrade_pack_path)

  AGV 软件升级。升级包路径，例如*”/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz”*。*10100000*：目标版本升级成功；*10100201*：正在升级中；*else*：升级失败。
- std::vector< std::string > agvc_interface::AgvcInterface::getSoftwareVersionList ()

  获取*AGV* 软件版本列表。软件版本列表集合 *{0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}*。
- int agvc_interface::AgvcInterface::switchSoftwareVersion (const std::string &software_version)

  切换*AGV* 软件版本。软件版本，例如*”0.3.2+3b807d1-Linux_x86_64”*。*10100000*：软件版本切换成功；*else*：软件版本切换失败。
- int agvc_interface::AgvcInterface::uninstallSoftware (const std::string &software_version)

  卸载*AGV* 软件版本。软件版本，例如*”0.3.2+3b807d1-Linux_x86_64”*。*10100000*：软件版本卸载成功；*else*：软件版本卸载失败。
- int agvc_interface::AgvcInterface::updateFirmware (const FirmwareUpdateParam &update_firmware)

  AGV 固件更新。配置要更新固件的模块、固件存储路径。*10100000*：下发固件更新信息成功；*else*：固件正在更新中。
- FirmwareUpdateProcessInfo agvc_interface::AgvcInterface::getUpdateFirmwareProcess ()

获取固件更新过程信息。固件更新步骤信息 *(*步骤信息为 *failed* 代表更新失败*)*，更新进度 *(*进度信息为 *1.0* 代表更新成功*)*。

- int agvc_interface::AgvcInterface::startCollectCalibrationData (const CalibrationType &type, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

  开始采集数据。*LASER_ODOM* 标定采集数据量要求大于 *1000* 组，具体数据量可以在配置文件中修改。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。

- int agvc_interface::AgvcInterface::cancelCollectCalibrationData (const CalibrationType &type, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

  取消采集数据。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。

- int agvc_interface::AgvcInterface::startCalibration (const CalibrationType &type, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

  开始标定。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。

- CalibrationProcessInfo agvc_interface::AgvcInterface::getCalibrationProcessInfo (const CalibrationType &type)

  获取标定信息。标定类型。标定信息。

- int agvc_interface::AgvcInterface::restartAgv ()

  重启*AGV*。*10100000*：重启指令执行成功；*else*：重启指令执行失败。

- int agvc_interface::AgvcInterface::setAgvName (const std::string &agv_name)

  设置*AGV* 的名称。设置的*AGV* 名称。*10100000*：设置名称成功；*else*：设置名称失败。

- int agvc_interface::AgvcInterface::setPriority (const std::string &name, const std::string &ip=””)

  设置控制权。*1.*

- int agvc_interface::AgvcInterface::releasePriority ()

  释放控制权。*10100000*：释放控制权成功；*else*：释放控制权失败。

- int agvc_interface::AgvcInterface::scriptPaused ()

  暂停脚本。*10100000*：脚本暂停成功；*else*：脚本暂停失败。

- int agvc_interface::AgvcInterface::scriptResume ()

  恢复脚本。*10100000*：脚本恢复成功；*else*：脚本恢复失败。

- int agvc_interface::AgvcInterface::scriptStop ()

  停止脚本。*10100000*：脚本停止成功；*else*：脚本停止失败。

- int agvc_interface::AgvcInterface::setScriptStatus (ScriptRuntimeState status)

  设置脚本运行状态。*10100000*：设置脚本运行状态成功；*else*：设置脚本运行状态失败。

- int agvc_interface::AgvcInterface::autoAlignRailway (const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

9.7.1 详细描述

AGVC 系统配置、控制权、软件升级、固件更新等系统级功能

9.7.2 函数说明

autoAlignRailway()

```
int agvc_interface::AgvcInterface::autoAlignRailway (
          const Header & command_header = { "99999", "99999", 1, "99999" })
```

注意

1. 异步接口；2.控制权限制。自动对接轨道并上轨。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：自动上轨成功；10100201：正在上轨中；10120202：无控制权；else：上轨失败。

cancelCollectCalibrationData()

```
int agvc_interface::AgvcInterface::cancelCollectCalibrationData (
          const CalibrationType & type,
          const Header & command_header = { "99999", "99999", 1, "99999" })
```

取消采集数据。标定类型。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：接口调用成功；10120202：无控制权；else：接口调用失败。

connectWifi()

```
int agvc_interface::AgvcInterface::connectWifi (
          const std::string & ssid,
          const std::string & password)
```

连接到指定WiFi，自动切换到WiFi 模式。必须在有线连接时调用，且执行时间较长。WiFi 名称。WiFi 密码。10100000：连接WiFi 成功；else：连接WiFi 失败。

enableHotspot()

```
int agvc_interface::AgvcInterface::enableHotspot (
          const std::string & password)
```

开启热点，自动切换到热点模式。必须在有线连接时调用，且执行时间较长。热点密码；如果为空使用”administrator” 为默认密码，SSID 默认为SN 码。10100000：启用热点成功；else：启用热点失败。

getAgvControllerParametersFile()

```
std::string agvc_interface::AgvcInterface::getAgvControllerParametersFile ()
```

获取AGV 控制器的参数文件。当前AGV 控制器中的参数信息。

getAsyncInterfaceResultStatus()

```
AsyncInterfaceResultStatus agvc_interface::AgvcInterface::getAsyncInterfaceResultStatus ()
```

查询异步接口的运行情况，不代表异步接口本身的运行结果。异步接口：saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv。异步接口：getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv。异步接口：getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv。异步接口：autoAlignRailway。客户端与AGV 断联后，可调用该接口获取异步接口运行情况。返回值的每个字段中均有 header，其中 header.id 表示下发异步任务时的 id 号。NONE：异步接口未运行；RUNNING：异步接口运行中；FINISH：异步接口运行结束。

getCalibrationProcessInfo()

```
CalibrationProcessInfo agvc_interface::AgvcInterface::getCalibrationProcessInfo (
          const CalibrationType & type)
```

获取标定信息。标定类型。标定信息。

getIpAddressList()

```
std::vector< IpAddressInfo > agvc_interface::AgvcInterface::getIpAddressList ()
```

获取本机所有IP 地址。本机所有IP 地址。

getSoftwareVersionList()

```
std::vector< std::string > agvc_interface::AgvcInterface::getSoftwareVersionList ()
```

获取AGV 软件版本列表。软件版本列表集合 {0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}。

getUpdateFirmwareProcess()

```
FirmwareUpdateProcessInfo agvc_interface::AgvcInterface::getUpdateFirmwareProcess ()
```

获取固件更新过程信息。固件更新步骤信息 (步骤信息为 failed 代表更新失败)，更新进度 (进度信息为 1.0 代表更新成功)。

getWifiList()

```
std::vector< std::string > agvc_interface::AgvcInterface::getWifiList ()
```

注意

异步接口。获取当前扫描到的WiFi 列表。{RUNNING}: 异步接口运行中; {}: 为空, 未扫描到WIFI; {SSID}：不为空，扫描到的WIFI 名称列表。

refreshAgvControllerParametersFile()

```
int agvc_interface::AgvcInterface::refreshAgvControllerParametersFile ()
```

注意

控制权限制。修改AGV 控制器的参数文件后，让AGV 各节点刷新参数。不要高频调用，调用频率低于 1Hz。10100000：接口执行成功；else：接口执行失败。

releasePriority()

```
int agvc_interface::AgvcInterface::releasePriority ()
```

释放控制权。10100000：释放控制权成功；else：释放控制权失败。

resetAgvControllerParametersFile()

```
int agvc_interface::AgvcInterface::resetAgvControllerParametersFile ()
```

注意

控制权限制。将AGV 控制器的参数文件恢复出厂设置。不要高频调用, 调用频率低于 1Hz。10100000：接口执行成功；else：接口执行失败。

restartAgv()

```
int agvc_interface::AgvcInterface::restartAgv ()
```

重启AGV。10100000：重启指令执行成功；else：重启指令执行失败。

scriptPaused()

```
int agvc_interface::AgvcInterface::scriptPaused ()
```

暂停脚本。10100000：脚本暂停成功；else：脚本暂停失败。

scriptResume()

```
int agvc_interface::AgvcInterface::scriptResume ()
```

恢复脚本。10100000：脚本恢复成功；else：脚本恢复失败。

scriptStop()

```
int agvc_interface::AgvcInterface::scriptStop ()
```

停止脚本。10100000：脚本停止成功；else：脚本停止失败。

setAgvControllerParametersFile()

```
int agvc_interface::AgvcInterface::setAgvControllerParametersFile (
            const std::string & agv_parameters)
```

注意

控制权限制。设置AGV 控制器的参数文件。不要高频调用，调用频率低于 1Hz；此接口会默认执行刷新参数功能。AGV 控制器的参数信息。10100000：设置参数成功；else：设置参数失败。

setAgvName()

```
int agvc_interface::AgvcInterface::setAgvName (
            const std::string & agv_name)
```

设置AGV 的名称。设置的AGV 名称。10100000：设置名称成功；else：设置名称失败。

setPriority()

```
int agvc_interface::AgvcInterface::setPriority (
            const std::string & name,
            const std::string & ip = "")
```

设置控制权。1.
该接口设置成功后才能正常使用控制权相关接口；2.控制权设置前需要先释放他人的控制权：release↩Priority；3.Web 端抢占优先权：name=”web”。登录RPC 注册时的用户名。IP 地址。10100000：设置控制权成功；else：设置控制权失败。

setScriptStatus()

```
int agvc_interface::AgvcInterface::setScriptStatus (
            ScriptRuntimeState status)
```

设置脚本运行状态。10100000：设置脚本运行状态成功；else：设置脚本运行状态失败。

setSystemClock()

```
int agvc_interface::AgvcInterface::setSystemClock (
            const std::string & stamp)
```

设置AGV 系统时间。时间戳，格式为YYYY-MM-DDTHH:mm:ss.ssZ，例如“2017-04-15T11:40:03.12Z”。

startCalibration()

```
int agvc_interface::AgvcInterface::startCalibration (
            const CalibrationType & type,
            const Header & command_header = { "99999", "99999", 1, "99999" })
```

开始标定。标定类型。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：接口调用成功；10120202：无控制权；else：接口调用失败。

startCollectCalibrationData()

```
int agvc_interface::AgvcInterface::startCollectCalibrationData (
            const CalibrationType & type,
            const Header & command_header = { "99999", "99999", 1, "99999" })
```

开始采集数据。LASER_ODOM 标定采集数据量要求大于 1000 组，具体数据量可以在配置文件中修改。标定类型。id 字段代表下发本次命令的 id；其他字段无特殊含义。10100000：接口调用成功；10120202：无控制权；else：接口调用失败。

switchSoftwareVersion()

```
int agvc_interface::AgvcInterface::switchSoftwareVersion (
            const std::string & software_version)
```

切换AGV 软件版本。软件版本，例如”0.3.2+3b807d1-Linux_x86_64”。10100000：软件版本切换成功；else：软件版本切换失败。

uninstallSoftware()

```
int agvc_interface::AgvcInterface::uninstallSoftware (
            const std::string & software_version)
```

卸载AGV 软件版本。软件版本，例如”0.3.2+3b807d1-Linux_x86_64”。10100000：软件版本卸载成功；else：软件版本卸载失败。

updateFirmware()

```
int agvc_interface::AgvcInterface::updateFirmware (
        const FirmwareUpdateParam & update_firmware)
```

AGV 固件更新。配置要更新固件的模块、固件存储路径。10100000：下发固件更新信息成功；else：固件正在更新中。

updateSoftware()

```
int agvc_interface::AgvcInterface::updateSoftware (
        const std::string & upgrade_pack_path)
```

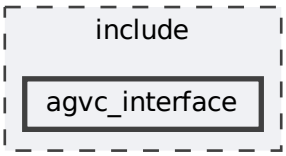
AGV 软件升级。升级包路径,例如"/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz"。10100000：目标版本升级成功；10100201：正在升级中；else：升级失败。

Chapter 10

目录说明

10.1 include/agvc_interface 目录参考

agvc_interface 的目录依赖关系图



文件

- 文件 agvc_interface.h
- 文件 rpc.h
  用于*RPC* 模块的交互，如登录、连接等功能
- 文件 rtde.h
  用于*RPC* 模块的交互，如订阅、发布等功能
- 文件 script.h
  用于*SCRIPT* 模块的交互，如向服务器发送脚本
- 文件 type.h

10.2 doc 目录参考

10.3 include 目录参考

目录

- 目录 agvc_interface

Chapter 11

命名空间文档

11.1 agvc_interface 命名空间参考

类

- class AgvcInterface
- class RpcClient
  RPC 客户端
- class OutputBuilder
  向输出数据中增加
- class InputParser
  解析输入
- class RtdeClient
  RTDE 客户端
- class ScriptWriter
- class ScriptClient
  SCRIPT 客户端
- struct Point2d
  二维点
- struct Pose2d
  二维位姿
- struct Speed
  速度
- struct Header
  头部信息
- struct AgvDetails
  agv 信息
- struct RunningInfo
  agv 运行信息
- struct SaveMapWorkingStatus
  接口 *saveMap* 的工作状态
- struct SwitchMapWorkingStatus
  接口 *switchMap* 的工作状态
- struct ChangeModeWorkingStatus
  接口 *changeRunningMode* 的工作状态
- struct RelocationWorkingStatus
  接口*Relocation* 的工作状态
- struct SetPngMapAllInfoWorkingStatus
  接口 *setPngMapAllInfo* 的工作状态
- struct GetPngMapAllInfoWorkingStatus

接口 *getPngMapAllInfo* 的工作状态

- struct GetGridMapWorkingStatus

  接口 *getGridMapFromAgv* 的工作状态
- struct SendGridMapWorkingStatus

  接口 *sendGridMapToAgv* 的工作状态
- struct GetPngMapWorkingStatus
- struct SendPngMapWorkingStatus

  接口 *sendBase64PngMapToAgv* 的工作状态
- struct SetGridMapAllInfoWorkingStatus

  接口 *setGridMapAllInfo* 的工作状态
- struct GetGridMapAllInfoWorkingStatus

  接口 *getGridMapAllInfo* 的工作状态
- struct PreviewPngMapWorkingStatus

  接口 *previewPngMapFromAgv* 的工作状态
- struct AutoAlignRailwayWorkingStatus

  接口 *autoAlignRailway* 的工作状态
- struct AsyncInterfaceResultStatus

  异步接口运行状态暂时存在 *14* 个异步接口: *saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMap↩FromAgv getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv, autoAlignRailway*
- struct StationMark

  站点
- struct PathStation

  通过站点表示路径
- struct MapInfo

  地图信息
- struct MapVirtualArea

  地图中的虚拟区域
- struct MapAllInfo

  包含当前地图、当前地图上的站点、当前地图上的路径、当前地图上的虚拟区域信息
- struct IpAddressInfo

  IP 地址
- struct AdjustParam

  agv 到点后二次调整位置参数设置
- struct NavGoalType

  使 *agv* 导航到目标位置（控制信息）
- struct NavInfo

  导航状态信息 / 自动充电状态信息
- struct AutoChargingCommand

  自动充电信息
- struct FirmwareUpdateParam

  配置要更新固件的模块及固件路径
- struct FirmwareUpdateProcessInfo

  固件更新过程信息
- struct CalibrationProcessInfo
- struct RtdeRecipe

  RTDE 菜单
- class AgvcException

类型定义

- using AgvcInterfacePtr = std::shared_ptr<AgvcInterface>
- using RpcClientPtr = std::shared_ptr<RpcClient>
- using RtdeClientPtr = std::shared_ptr<RtdeClient>
- using ScriptClientPtr = std::shared_ptr<ScriptClient>
- typedef FeedbackStatus NavStatus
  agv 导航状态
- typedef FeedbackStatus SaveMapStatus
  保存地图异步接口状态，调用接口 *saveMap* 后的状态
- typedef FeedbackStatus SwitchMapStatus
  切换地图异步接口状态，调用接口 *switchMap* 后的状态
- typedef FeedbackStatus ChangeRunningModeStatus
  切换运行模式异步接口状态，调用接口 *changeRunningMode* 后的状态
- typedef FeedbackStatus RelocationStatus
  重定位状态，调用接口 *relocation* 后的状态
- typedef FeedbackStatus SetPngMapAllInfoStatus
  上传*Png* 地图全部信息接口状态，调用接口 *setPngMapAllInfo* 后的状态
- typedef FeedbackStatus GetPngMapAllInfoStatus
  获取*Png* 地图全部信息接口状态，调用接口 *getPngMapAllInfo* 后的状态
- typedef FeedbackStatus GetGridMapStatus
  获取栅格地图接口状态，调用接口 *getGridMapFromAgv* 后的状态
- typedef FeedbackStatus SendGridMapStatus
  发送栅格地图接口状态，调用接口 *sendGridMapToAgv* 后的状态
- typedef FeedbackStatus GetPngMapStatus
  获取 *png* 地图接口状态，调用接口 *getBase64PngMapFromAgv* 后的状态
- typedef FeedbackStatus SendPngMapStatus
  发送 *png* 地图接口状态，调用接口 *sendBase64PngMapToAgv* 后的状态
- typedef FeedbackStatus SetGridMapAllInfoStatus
  设置栅格地图全部信息接口状态，调用接口 *setGridMapAllInfo* 后的状态
- typedef FeedbackStatus GetGridMapAllInfoStatus
  获取栅格地图全部信息接口状态，调用接口 *getGridMapAllInfo* 后的状态
- typedef FeedbackStatus PreviewPngMapStatus
  获取栅格地图全部信息接口状态，调用接口 *previewPngMapFromAgv* 后的状态
- typedef FeedbackStatus AutoAlignRailwayStatus
  获取自动上轨接口状态，调用接口 *autoAlignRailway* 后的状态
- typedef FeedbackStatus CalibrationStatus
  标定状态
- typedef MapInfo OccupancyGridMap
  栅格地图
- typedef MapInfo Base64PngMap
  Base64 编码的 *png* 地图图片

枚举

- enum AgvcErrorCodes : int { ENUM_AgvcErrorCodes_DECLARES }
- enum class RunningMode {
  NONE = 0 , START = 1 , MAPPING = 2 , NAVIGATE = 3 ,
  CHARGING = 4 , MAINTAIN = 5 , STOP = 6 }
  agv 运行模式
- enum class PathShape { NONE = 0 , LINE = 1 , ARC = 2 , BEZIER = 3 }
  路径的形状
- enum class MapFormat { NONE = 0 , OCCUPANCY_GRID = 1 , BASE64_PNG = 2 }

地图格式

- enum class MapVirtualAreaType { NONE = 0 , OBSTACLE = 1 , SLOW_DOWN = 2 , INFLATE = 3 }

  地图中虚拟区域的类型

- enum class MapVirtualAreaShape { NONE = 0 , LINE = 1 , ARC = 2 , CIRCLE = 3 , POLYGON = 4 }

  地图中虚拟区域的形状

- enum class NavType { NONE = 0 , FREE_TO_POSE = 1 , FREE_TO_STATION = 2 , PATH_TO_STATION = 3 }

  agv 导航类型

- enum class AutoChargingType { NONE = 0 , LOW_POWER_TO_BOARD = 1 , HIGH_POWER_LEAVE_BOARD = 2 , FORCE_TO_BOARD = 3 , FORCE_LEAVE_BOARD = 4 }

  自动充电类型

- enum class FeedbackStatus { NONE = 0 , FINISHED = 1 , FAILED = 2 , RUNNING = 3 , PAUSED = 4 , CANCELED = 5 }

  一个操作执行后反馈的状态

- enum class FirmwareUpdateMode { NONE = 0 , UPDATE = 1 , FORCED_UPDATE = 2 , NOT_UPDATE = 3 }

  模块固件更新配置

- enum class ScriptRuntimeState { RUNNING = 0 , PAUSED = 1 , STOPPED = 2 , ABORTING = 3 }
- enum class CalibrationType { NONE = 0 , LASER_ODOM = 1 }

  标定类型

- enum error_type { parse_error = -32700 , invalid_request = -32600 , method_not_found = -32601 , invalid_params = -32602 , internal_error = -32603 , server_error , invalid }
- enum ExceptionCode { EC_DISCONNECTED = -1 , EC_NOT_LOGINED = -2 , EC_INVAL_SOCKET = -3 , EC_REQUEST_BUSY = -4 , EC_SEND_FAILED = -5 , EC_RECV_TIMEOUT = -6 , EC_RECV_ERROR = -7 , EC_PARSE_ERROR = -8 , EC_INVALID_REQUEST = -9 , EC_METHOD_NOT_FOUND = -10 , EC_INVALID_PARAMS = -11 , EC_INTERNAL_ERROR = -12 , EC_SERVER_ERROR = -13 , EC_INVALID = -14 }

函数

- const char * returnValue2Str (int retval)

11.1.1 类型定义说明

AgvcInterfacePtr

```
using agvc_interface::AgvcInterfacePtr = std::shared_ptr<AgvcInterface>
```

在文件 agvc_interface.h 第 1492 行定义.

AutoAlignRailwayStatus

typedef FeedbackStatus agvc_interface::AutoAlignRailwayStatus
获取自动上轨接口状态，调用接口 autoAlignRailway 后的状态
NONE = 0, // 自动上轨接口未执行
FINISHED = 1, // 自动上轨接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 自动上轨接口执行中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 380 行定义.

Base64PngMap

typedef MapInfo agvc_interface::Base64PngMap
Base64 编码的 png 地图图片
Header header; // 地图 id，地图名称，当前时间，map_id 为地图的唯一 id
double resolution; // 分辨率，每个正方形像素表示的宽度，单位 m
uint32_t width; // 地图宽度，多少个像素宽度
uint32_t height; // 地图高度，多少个像素高度
Pose2d origin; // {x(m) y(m) theta(rad)} // 地图原点在真实地图下的坐标，指地图原点对应的实际位置坐标。// 一张地图中，左下角为地图原点；
MapFormat format; // 地图格式，=MapFormat::BASE64_PNG
std::vector<int8_t> data; // 地图数据信息；Base64 编码的 png 地图图片，可以将 data 类型转换为 string 类型使用
在文件 type.h 第 706 行定义.

CalibrationStatus

typedef FeedbackStatus agvc_interface::CalibrationStatus
标定状态
NONE = 0, // 空
FINISHED = 1, // 标定完成
FAILED = 2, // 标定失败
RUNNING = 3, // 正在采集数据
PAUSED = 4, // 无效值
CANCELED = 5, // 取消标定
在文件 type.h 第 392 行定义.

ChangeRunningModeStatus

typedef FeedbackStatus agvc_interface::ChangeRunningModeStatus
切换运行模式异步接口状态，调用接口 changeRunningMode 后的状态
NONE = 0, // 切换运行模式接口未执行
FINISHED = 1, // 切换运行模式接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 切换运行模式中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 248 行定义.

GetGridMapAllInfoStatus

typedef FeedbackStatus agvc_interface::GetGridMapAllInfoStatus
获取栅格地图全部信息接口状态，调用接口 getGridMapAllInfo 后的状态
NONE = 0, // 获取栅格地图全部信息接口未执行
FINISHED = 1, // 获取栅格地图全部信息接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 栅格地图全部信息获取中

PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 356 行定义.

GetGridMapStatus

typedef FeedbackStatus agvc_interface::GetGridMapStatus
获取栅格地图接口状态，调用接口 getGridMapFromAgv 后的状态
NONE = 0, // 获取栅格地图接口未执行
FINISHED = 1, // 获取栅格地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 栅格地图获取中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 296 行定义.

GetPngMapAllInfoStatus

typedef FeedbackStatus agvc_interface::GetPngMapAllInfoStatus
获取Png 地图全部信息接口状态，调用接口 getPngMapAllInfo 后的状态
NONE = 0, // 获取地图信息接口未执行
FINISHED = 1, // 获取地图信息接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 地图信息获取中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 284 行定义.

GetPngMapStatus

typedef FeedbackStatus agvc_interface::GetPngMapStatus
获取 png 地图接口状态，调用接口 getBase64PngMapFromAgv 后的状态
NONE = 0, // 获取 png 地图接口未执行
FINISHED = 1, // 获取 png 地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 获取 png 地图中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 320 行定义.

NavStatus

typedef FeedbackStatus agvc_interface::NavStatus
agv 导航状态
NONE = 0, // 空
FINISHED = 1, // 向站点导航完成
FAILED = 2, // 向站点导航失败
RUNNING = 3, // 正在向站点导航
PAUSED = 4, // 向站点导航暂停
CANCELED = 5, // 向站点导航取消
在文件 type.h 第 212 行定义.

OccupancyGridMap

typedef MapInfo agvc_interface::OccupancyGridMap
栅格地图
Header header; // 地图 id，地图名称，当前时间，map_id 为地图的唯一 id
double resolution; // 分辨率，每个正方形栅格的宽度，单位 m
uint32_t width; // 地图宽度，多少个栅格宽度
uint32_t height; // 地图高度，多少个栅格高度

Pose2d origin; // {x(m) y(m) theta(rad)} // 栅格地图原点在真实地图下的坐标，指栅格原点对应的实际位置坐标。// 一张栅格地图中，左下角为栅格地图原点;
MapFormat format; // 地图格式，=MapFormat::OCCUPANCY_GRID
std::vector<int8_t> data; // 地图数据信息；0 表示可通行，100 表示被占用，-1 表示未知
在文件 type.h 第 691 行定义.

PreviewPngMapStatus

typedef FeedbackStatus agvc_interface::PreviewPngMapStatus
获取栅格地图全部信息接口状态，调用接口 previewPngMapFromAgv 后的状态
NONE = 0, // 预览地图接口未执行
FINISHED = 1, // 预览地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 预览地图接口获取中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 368 行定义.

RelocationStatus

typedef FeedbackStatus agvc_interface::RelocationStatus
重定位状态，调用接口 relocation 后的状态
NONE = 0, // 重定位接口未执行
FINISHED = 1, // 重定位接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 重定位中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 260 行定义.

RpcClientPtr

using agvc_interface::RpcClientPtr = std::shared_ptr<RpcClient>
在文件 rpc.h 第 147 行定义.

RtdeClientPtr

using agvc_interface::RtdeClientPtr = std::shared_ptr<RtdeClient>
在文件 rtde.h 第 284 行定义.

SaveMapStatus

typedef FeedbackStatus agvc_interface::SaveMapStatus
保存地图异步接口状态，调用接口 saveMap 后的状态
NONE = 0, // 保存地图接口未执行
FINISHED = 1, // 保存地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 保存地图中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 224 行定义.

ScriptClientPtr

using agvc_interface::ScriptClientPtr = std::shared_ptr<ScriptClient>
在文件 script.h 第 188 行定义.

SendGridMapStatus

typedef FeedbackStatus agvc_interface::SendGridMapStatus
发送栅格地图接口状态，调用接口 sendGridMapToAgv 后的状态
NONE = 0, // 发送栅格地图接口未执行
FINISHED = 1, // 发送栅格地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 发送栅格地图中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 308 行定义.

SendPngMapStatus

typedef FeedbackStatus agvc_interface::SendPngMapStatus
发送 png 地图接口状态，调用接口 sendBase64PngMapToAgv 后的状态
NONE = 0, // 发送 png 地图接口未执行
FINISHED = 1, // 发送 png 地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // png 地图发送中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 332 行定义.

SetGridMapAllInfoStatus

typedef FeedbackStatus agvc_interface::SetGridMapAllInfoStatus
设置栅格地图全部信息接口状态，调用接口 setGridMapAllInfo 后的状态
NONE = 0, // 发送栅格地图全部信息接口未执行
FINISHED = 1, // 发送栅格地图全部信息接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 栅格地图发送中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 344 行定义.

SetPngMapAllInfoStatus

typedef FeedbackStatus agvc_interface::SetPngMapAllInfoStatus
上传Png 地图全部信息接口状态，调用接口 setPngMapAllInfo 后的状态
NONE = 0, // 设置地图信息接口未执行
FINISHED = 1, // 设置地图信息接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 地图信息上传中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 272 行定义.

SwitchMapStatus

typedef FeedbackStatus agvc_interface::SwitchMapStatus
切换地图异步接口状态，调用接口 switchMap 后的状态
NONE = 0, // 切换地图接口未执行
FINISHED = 1, // 切换地图接口执行完毕
FAILED = 2, // 无效值
RUNNING = 3, // 切换地图中
PAUSED = 4, // 无效值
CANCELED = 5, // 无效值
在文件 type.h 第 236 行定义.

11.1.2 枚举类型说明

AgvcErrorCodes

```
enum agvc_interface::AgvcErrorCodes : int
```

枚举值

ENUM_AgvcErrorCodes_DECLARES	

在文件 type.h 第 71 行定义.

AutoChargingType

```
enum class agvc_interface::AutoChargingType [strong]
```
自动充电类型

自从

0.7.0

注解

高电量阈值与低电量阈值在对外配置参数中进行修改

枚举值

NONE	空
LOW_POWER_TO_BOARD	低电量自动上桩；如果当前电量低于低电量阈值，开始上桩；否则，不执行上桩；
HIGH_POWER_LEAVE_BOARD	高电量自动下桩；如果当前电量高于高电量阈值，开始下桩；否则，不执行下桩；
FORCE_TO_BOARD	强制自动上桩；无视当前电量
FORCE_LEAVE_BOARD	强制自动下桩；无视当前电量

在文件 type.h 第 151 行定义.

CalibrationType

```
enum class agvc_interface::CalibrationType [strong]
```
标定类型

枚举值

NONE	空
LASER_ODOM	激光-码盘里程计标定

在文件 type.h 第 196 行定义.

error_type

```
enum agvc_interface::error_type
```

枚举值

parse_error	解析错误
invalid_request	无效请求
method_not_found	方法未找到
invalid_params	无效参数
internal_error	内部错误
server_error	服务器错误
invalid	无效

在文件 type.h 第 850 行定义.

ExceptionCode

enum agvc_interface::ExceptionCode

枚举值

EC_DISCONNECTED	断开连接
EC_NOT_LOGINED	未登录
EC_INVAL_SOCKET	无效套接字
EC_REQUEST_BUSY	请求繁忙
EC_SEND_FAILED	发送失败
EC_RECV_TIMEOUT	接收超时
EC_RECV_ERROR	接收错误
EC_PARSE_ERROR	解析错误
EC_INVALID_REQUEST	无效请求
EC_METHOD_NOT_FOUND	方法未找到
EC_INVALID_PARAMS	无效参数
EC_INTERNAL_ERROR	内部错误
EC_SERVER_ERROR	服务器错误
EC_INVALID	无效

在文件 type.h 第 861 行定义.

FeedbackStatus

enum class agvc_interface::FeedbackStatus [strong]
一个操作执行后反馈的状态

枚举值

NONE	空
FINISHED	成功
FAILED	失败
RUNNING	正在运行

PAUSED	暂停
CANCELED	取消

在文件 type.h 第 163 行定义.

FirmwareUpdateMode

enum class agvc_interface::FirmwareUpdateMode [strong]
模块固件更新配置

枚举值

NONE	空默认不更新固件
UPDATE	更新固件
FORCED_UPDATE	强制更新固件
NOT_UPDATE	不更新固件

在文件 type.h 第 176 行定义.

MapFormat

enum class agvc_interface::MapFormat [strong]
地图格式

枚举值

NONE	
OCCUPANCY_GRID	占用栅格地图
BASE64_PNG	base64 编码的 png 地图

在文件 type.h 第 105 行定义.

MapVirtualAreaShape

enum class agvc_interface::MapVirtualAreaShape [strong]
地图中虚拟区域的形状

枚举值

NONE	
LINE	直线
ARC	圆弧
CIRCLE	圆形
POLYGON	多边形

在文件 type.h 第 126 行定义.

MapVirtualAreaType

enum class agvc_interface::MapVirtualAreaType [strong]
地图中虚拟区域的类型

枚举值

NONE	
OBSTACLE	虚拟墙
SLOW_DOWN	减速区
INFLATE	膨胀区

在文件 type.h 第 115 行定义.

NavType

enum class agvc_interface::NavType [strong]
agv 导航类型

枚举值

NONE	空
FREE_TO_POSE	自由导航到任意点
FREE_TO_STATION	自由导航到站点
PATH_TO_STATION	路径导航到站点

在文件 type.h 第 138 行定义.

PathShape

enum class agvc_interface::PathShape [strong]
路径的形状

枚举值

NONE	
LINE	直线
ARC	圆弧
BEZIER	贝塞尔曲线 B_SPLINE = 4 // B 样条曲线

在文件 type.h 第 93 行定义.

RunningMode

enum class agvc_interface::RunningMode [strong]
agv 运行模式

枚举值

NONE	

START	开始模式
MAPPING	建图模式
NAVIGATE	导航模式
CHARGING	自动充电模式
MAINTAIN	维护模式
STOP	停止模式

在文件 type.h 第 79 行定义.

ScriptRuntimeState

```
enum class agvc_interface::ScriptRuntimeState  [strong]
```

枚举值

RUNNING	
PAUSED	
STOPPED	
ABORTING	

在文件 type.h 第 185 行定义.

11.1.3 函数说明

returnValue2Str()

```
const char * agvc_interface::returnValue2Str (
          int retval)  [inline]
```

在文件 type.h 第 909 行定义.
引用了 ENUM_AgvcErrorCodes_DECLARES.

Chapter 12

类说明

12.1 agvc_interface::AdjustParam 结构体参考

agv 到点后二次调整位置参数设置
`#include <type.h>`

Public 属性

- bool enable

  是否要二次调整 *agv* 位置，*true*-调整；*false* 不调整
- bool forward_flag

  agv 调整方向，*true*，向车头方向调整，*false*，向车尾方向调整
- double max_distance

  agv 单次调整的最远距离
- double max_angle

  agv 单次调整的最大角度

12.1.1 详细描述

agv 到点后二次调整位置参数设置
在文件 type.h 第 752 行定义.

12.1.2 类成员变量说明

enable

`bool agvc_interface::AdjustParam::enable`
是否要二次调整 agv 位置，true-调整；false 不调整
在文件 type.h 第 754 行定义.

forward_flag

`bool agvc_interface::AdjustParam::forward_flag`
agv 调整方向，true，向车头方向调整，false，向车尾方向调整
在文件 type.h 第 755 行定义.

max_angle

`double agvc_interface::AdjustParam::max_angle`
agv 单次调整的最大角度
在文件 type.h 第 757 行定义.

max_distance

```
double agvc_interface::AdjustParam::max_distance
```

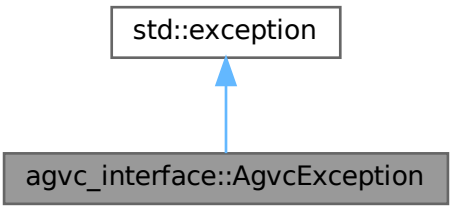
agv 单次调整的最远距离
在文件 type.h 第 756 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

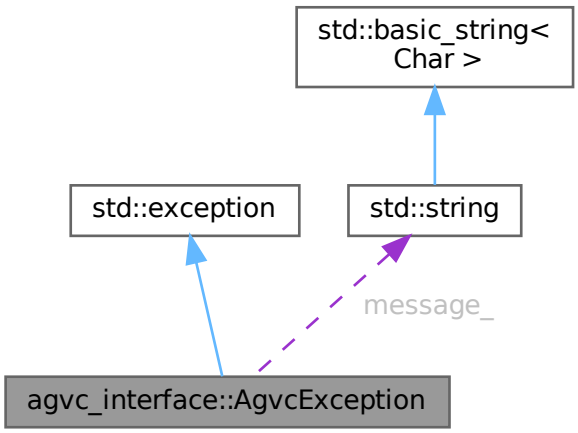
12.2 agvc_interface::AgvcException 类参考

```
#include <type.h>
```

类 agvc_interface::AgvcException 继承关系图:



agvc_interface::AgvcException 的协作图:



Public 成员函数

- AgvcException (int code, const std::string &prefix, const std::string &message) noexcept
- AgvcException (int code, const std::string &message) noexcept
- error_type type () const
- int code () const
- const char * what () const noexcept override

Private 属性

- int code_
- std::string message_

12.2.1 详细描述

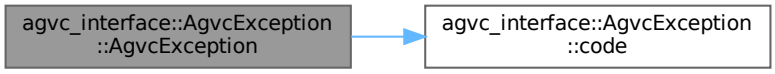
在文件 type.h 第 879 行定义.

12.2.2 构造及析构函数说明

AgvcException() [1/2]

```
agvc_interface::AgvcException::AgvcException (
            int code,
            const std::string & prefix,
            const std::string & message)  [inline], [noexcept]
```

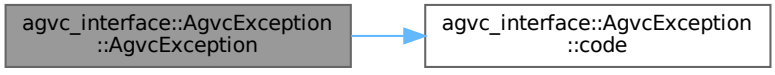
在文件 type.h 第 882 行定义.
引用了 code().
函数调用图:



AgvcException() [2/2]

```
agvc_interface::AgvcException::AgvcException (
            int code,
            const std::string & message)  [inline], [noexcept]
```

在文件 type.h 第 887 行定义.
引用了 code().
函数调用图:



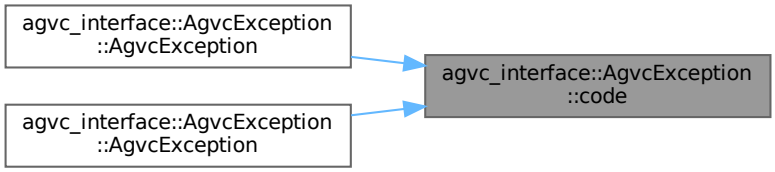
12.2.3 成员函数说明

code()

```
int agvc_interface::AgvcException::code () const  [inline]
```

在文件 type.h 第 901 行定义.
引用了 code_.
被这些函数引用 AgvcException() , 以及 AgvcException().

这是这个函数的调用关系图:



type()

```
error_type agvc_interface::AgvcException::type () const  [inline]
```

在文件 type.h 第 889 行定义.
引用了 code_, agvc_interface::invalid, agvc_interface::parse_error , 以及 agvc_interface::server_error.

what()

```
const char * agvc_interface::AgvcException::what () const  [inline], [override], [noexcept]
```

在文件 type.h 第 902 行定义.
引用了 message_.

12.2.4 类成员变量说明

code_

```
int agvc_interface::AgvcException::code_  [private]
```

在文件 type.h 第 905 行定义.
被这些函数引用 code() , 以及 type().

message_

```
std::string agvc_interface::AgvcException::message_  [private]
```

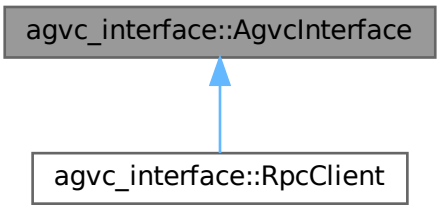
在文件 type.h 第 906 行定义.
被这些函数引用 what().
该类的文档由以下文件生成:

- include/agvc_interface/type.h

12.3 agvc_interface::AgvcInterface 类参考

```
#include <agvc_interface.h>
```

类 agvc_interface::AgvcInterface 继承关系图:



Public 成员函数

- AgvcInterface ()
- virtual ~AgvcInterface ()
- int setSystemClock (const std::string &stamp)

  设置*AGV* 系统时间。时间戳, 格式为*YYYY-MM-DDTHH:mm:ss.ssZ*, 例如 "*2017-04-15T11:40:03.12Z*"。
- AgvDetails getAgvDetails ()
- RunningInfo getRunningInfo ()
- Pose2d getAgvCurrentPose ()
- std::vector< Point2d > getLaserPointCloud ()
- AsyncInterfaceResultStatus getAsyncInterfaceResultStatus ()

  查询异步接口的运行情况, 不代表异步接口本身的运行结果。异步接口: *saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv*。异步接口: *getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv*。异步接口: *getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv*。异步接口: *autoAlignRailway*。客户端与*AGV* 断联后, 可调用该接口获取异步接口运行情况。返回值的每个字段中均有 *header*, 其中 *header.id* 表示下发异步任务时的 *id* 号。*NONE*: 异步接口未运行; *RUNNING*: 异步接口运行中; *FINISH*: 异步接口运行结束。
- std::vector< StationMark > getAllStations ()

  查询当前地图的所有站点信息。当前地图的所有站点信息。
- std::vector< StationMark > getAllStationsOfTargetMap (const Header &map_header)

  查询指定地图的所有站点信息。指定地图 *header(*命令 *id*, 名称, 时间, 地图 *id)*。指定地图的所有站点信息。
- int addStation (const StationMark &station)

  添加或修改一个站点。待添加或修改的站点信息。*10100000*: 添加或修改站点成功; *else*: 添加或修改站点失败。
- int addStations (const std::vector< StationMark > &stations)

  添加或修改多个站点。待添加或修改的站点信息。*10100000*: 添加或修改站点成功; *else*: 添加或修改站点失败。
- int addCPStationUseChargingBoard (const Header &station_header, const double &dis_station_board=1.5)

  通过自动识别充电桩位姿来添加充电站点。待添加或修改的充电站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。充电站点与充电桩之间的距离 *(*充电站点位于*AGV* 中心*)*, 单位: *m*, 范围*[1.0~2.0]*。*10100000*: 添加充电站点成功; *else*: 添加充电站点失败。
- int deleteStation (const Header &station_header)

  删除指定站点, 会删除与该站点相关的路径。待删除站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除站点成功; *else*: 删除站点失败。
- int deleteStations (const std::vector< Header > &stations_header)

  删除多个站点, 会删除与站点相关的路径。待删除站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除站点成功; *else*: 删除站点失败。

- std::vector< PathStation > getAllPaths ()

  查询当前地图的所有路径信息。当前地图的所有路径信息。
- std::vector< PathStation > getAllPathsOfTargetMap (const Header &map_header)

  查询指定地图的所有路径信息。指定地图 *header(*命令 *id*，名称，时间，地图 *id)*。指定地图的所有路径信息。
- PathStation getCurrentPath ()

  查询*AGV* 当前正在跟踪的路径信息。当前正在跟踪的路径信息。
- int generatePath (const PathStation &path_station)

  生成或修改一条路径。待生成或修改的路径信息。*10100000*：修改或生成路径成功；*else*：修改或生成路径失败。
- int generatePaths (const std::vector< PathStation > &paths_station)

  生成或修改多条路径。待生成或修改的路径信息。*10100000*：修改或生成路径成功；*else*：修改或生成路径失败。
- int deletePath (const Header &path_header)

  删除一条路径。待删除路径信息 *(*路径 *id*，名称，时间，地图 *id)*。*10100000*：删除路径成功；*else*：删除路径失败。
- int deletePaths (const std::vector< Header > &paths_header)

  删除多条路径。待删除路径信息 *(*路径 *id*，名称，时间，地图 *id)*。*10100000*：删除路径成功；*else*：删除路径失败。
- std::vector< Header > getMapList ()

  获取*AGV* 所有地图的信息头。所有地图的信息头 *(*命令 *id*，名称，时间，地图 *id)*。
- Header getCurrentMapHeader ()

  查询当前*AGV* 地图的信息头。当前地图的信息头 *(*命令 *id*，名称，时间，地图 *id)*。
- OccupancyGridMap getGridMapFromAgv (const Header &map_header)

  获取指定栅格地图信息。指定地图的信息头 *(*本次异步命令 *id*，名称，时间，地图 *id)*。*map_id* 为*"current_map"* 时，可特指为当前地图。指定栅格地图信息。
- Base64PngMap getBase64PngMapFromAgv (const Header &map_header)
- Base64PngMap previewPngMapFromAgv (const Header &map_header, const int &image_width_px=0, const int &image_height_px=0)
- int sendGridMapToAgv (const OccupancyGridMap &map)
- int sendBase64PngMapToAgv (const Base64PngMap &map)
- int saveMap (const Header &map_header)
- int switchMap (const Header &map_header)
- int deleteMap (const Header &map_header)

  删除一张指定地图。指定地图的信息头 *(*命令 *id*，名称，时间，地图 *id)*。*10100000*：删除地图成功；*else*：删除地图失败。
- int deleteMaps (const std::vector< Header > &map_headers)

  删除多张指定地图。多个指定地图的信息头 *(*命令 *id*，名称，时间，地图 *id)*。*10100000*：删除地图成功；*else*：删除地图失败。
- std::vector< MapVirtualArea > getAllMapVirtualArea ()

  查询当前地图的虚拟区域。当前地图的虚拟区域信息。
- std::vector< MapVirtualArea > getAllMapVirtualAreaOfTargetMap (const Header &map_header)

  查询目标地图的虚拟区域。指定地图 *header(*命令 *id*，名称，时间，地图 *id)*。指定地图的所有虚拟区域。
- int addMapVirtualArea (const MapVirtualArea &map_virtual_area)

  添加或修改一个虚拟区域。待添加或修改的虚拟区域信息。*10100000*：添加或修改虚拟区域成功；*else*：添加或修改虚拟区域失败。
- int addMapVirtualAreas (const std::vector< MapVirtualArea > &map_virtual_areas)

  添加或修改多个虚拟区域。待添加或修改的虚拟区域信息。*10100000*：添加或修改虚拟区域成功；*else*：添加或修改虚拟区域失败。
- int deleteMapVirtualArea (const Header &virtual_area_header)

  删除指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*，虚拟区域名称，时间，所在地图 *id)*。*10100000*：删除虚拟区域成功；*else*：删除虚拟区域失败。
- int deleteMapVirtualAreas (const std::vector< Header > &virtual_areas_header)

删除多个指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*，虚拟区域名称，时间，所在地图 *id)*。*10100000*: 删除虚拟区域成功；*else*: 删除虚拟区域失败。

- MapAllInfo getGridMapAllInfo (const Header &map_header)
- MapAllInfo getPngMapAllInfo (const Header &map_header)
- int setGridMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ "99999", "99999", 1, "99999" })
- int setPngMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ "99999", "99999", 1, "99999" })
- int deleteMapAllInfo (const Header &map_header)

  删除指定地图的全部信息，包括地图数据、站点、路径、虚拟区信息。指定地图信息头 *(*命令 *id*，地图名称，时间，地图 *id)*。*10100000*: 删除成功；*else*: 删除失败。
- std::vector< std::string > getWifiList ()
- int connectWifi (const std::string &ssid, const std::string &password)

  连接到指定 *WiFi*，自动切换到 *WiFi* 模式。必须在有线连接时调用，且执行时间较长。*WiFi* 名称。*WiFi* 密码。*10100000*: 连接 *WiFi* 成功；*else*: 连接 *WiFi* 失败。
- int enableHotspot (const std::string &password)

  开启热点，自动切换到热点模式。必须在有线连接时调用，且执行时间较长。热点密码；如果为空使用 *"administrator"* 为默认密码，*SSID* 默认为 *SN* 码。*10100000*: 启用热点成功；*else*: 启用热点失败。
- std::vector< IpAddressInfo > getIpAddressList ()

  获取本机所有 *IP* 地址。本机所有 *IP* 地址。
- int changeRunningMode (const RunningMode &running_mode, const Header &command_header={ "99999", "99999", 1, "99999" })
- int setControlSpeed (const Speed &speed)
- int setNavGoal (const NavGoalType &target)
- NavInfo getNavInfo ()
- int pauseAgvSpeed ()
- int resumeAgvSpeed ()
- int cancelNavigation ()
- int relocation (const Pose2d &init_pose, const Header &command_header={ "99999", "99999", 1, "99999" })
- std::string getAgvControllerParametersFile ()

  获取 *AGV* 控制器的参数文件。当前 *AGV* 控制器中的参数信息。
- int setAgvControllerParametersFile (const std::string &agv_parameters)
- int refreshAgvControllerParametersFile ()
- int resetAgvControllerParametersFile ()
- int checkPowerAndAutoCharge (const Header &check_power_header={ "99999", "99999", 99999, "99999" }, const std::string &nav_board_charge_station_id="99999")
- int setLeaveBoardTargetStation (const Header &leave_board_target_station_header)

  设置自动下桩时的目标站点。自动下桩的站点 *header* 信息。*10100000*: 设置自动下桩站点成功；*else*: 设置自动下桩站点失败。
- Header getLeaveBoardTargetStation ()

  获取自动下桩时的目标站点。自动下桩时的目标站点 *Header* 信息。
- int forcedAgvToChargingBoard (const Header &forced_charge_header={ "99999", "99999", 99999, "99999" }, const std::string &nav_board_charge_station_id="99999")
- int forcedAgvLeaveChargingBoard (const Header &forced_leave_header={ "99999", "99999", 99999, "99999" }, const std::string &leave_board_target_station_id="99999")
- int sendAutoChargingCommand (const AutoChargingCommand &auto_charging_command)
- std::string getAgvLogMessage ()

  获取 *AGV* 运行中的日志信息。*AGV* 运行中的日志信息。
- int updateSoftware (const std::string &upgrade_pack_path)

  AGV 软件升级。升级包路径，例如 *"/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz"*。*10100000*: 目标版本升级成功；*10100201*: 正在升级中；*else*: 升级失败。
- std::vector< std::string > getSoftwareVersionList ()

  获取 *AGV* 软件版本列表。软件版本列表集合 *{0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}*。

- int switchSoftwareVersion (const std::string &software_version)

  切换*AGV* 软件版本。软件版本，例如*"0.3.2+3b807d1-Linux_x86_64"*。*10100000*：软件版本切换成功；*else*：软件版本切换失败。
- int uninstallSoftware (const std::string &software_version)

  卸载*AGV* 软件版本。软件版本，例如*"0.3.2+3b807d1-Linux_x86_64"*。*10100000*：软件版本卸载成功；*else*：软件版本卸载失败。
- int updateFirmware (const FirmwareUpdateParam &update_firmware)

  AGV 固件更新。配置要更新固件的模块、固件存储路径。*10100000*：下发固件更新信息成功；*else*：固件正在更新中。
- FirmwareUpdateProcessInfo getUpdateFirmwareProcess ()

  获取固件更新过程信息。固件更新步骤信息 *(*步骤信息为 *failed* 代表更新失败*)*，更新进度 *(*进度信息为 *1.0* 代表更新成功*)*。
- int startCollectCalibrationData (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  开始采集数据。*LASER_ODOM* 标定采集数据量要求大于 *1000* 组，具体数据量可以在配置文件中修改。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。
- int cancelCollectCalibrationData (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  取消采集数据。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。
- int startCalibration (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  开始标定。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*：接口调用成功；*10120202*：无控制权；*else*：接口调用失败。
- CalibrationProcessInfo getCalibrationProcessInfo (const CalibrationType &type)

  获取标定信息。标定类型。标定信息。
- int restartAgv ()

  重启*AGV*。*10100000*：重启指令执行成功；*else*：重启指令执行失败。
- int setAgvName (const std::string &agv_name)

  设置*AGV* 的名称。设置的*AGV* 名称。*10100000*：设置名称成功；*else*：设置名称失败。
- int setPriority (const std::string &name, const std::string &ip="")

  设置控制权。*1.*
- int releasePriority ()

  释放控制权。*10100000*：释放控制权成功；*else*：释放控制权失败。
- int scriptPaused ()

  暂停脚本。*10100000*：脚本暂停成功；*else*：脚本暂停失败。
- int scriptResume ()

  恢复脚本。*10100000*：脚本恢复成功；*else*：脚本恢复失败。
- int scriptStop ()

  停止脚本。*10100000*：脚本停止成功；*else*：脚本停止失败。
- ScriptRuntimeState getScriptStatus ()

  获取脚本运行状态。脚本运行状态。
- int setScriptStatus (ScriptRuntimeState status)

  设置脚本运行状态。*10100000*：设置脚本运行状态成功；*else*：设置脚本运行状态失败。
- std::string errorCodeDecoder (const int &error_code)

  错误码查询。错误码。错误码含义；为空表示错误码未知。
- std::vector< Header > getCurrentErrorCodes ()

  获取当前的错误码。错误码集合。
- int soundLightPrompt ()

  寻找*AGV*，自动播放语音并特殊灯光闪烁。*10100000*：寻找指令下发成功；*else*：下发指令失败。
- bool isObstacleAhead (const double &detect_distance=1.0)

AGV 行进方向是否有障碍物。检测距离，默认为 *1.0m*。*true*：行进方向存在障碍物；*false*：行进方向无障碍物。

- StationMark getNearestStation (const double &detect_distance=1.0)

  获取指定检测距离范围内的最近站点信息。检测距离，默认为 *1.0m*。最近站点信息，*id* 为*NONE* 代表检测距离内无站点。

- int autoAlignRailway (const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

12.3.1 详细描述

在文件 agvc_interface.h 第 98 行定义.

12.3.2 构造及析构函数说明

AgvcInterface()

```
agvc_interface::AgvcInterface::AgvcInterface ()
```

~AgvcInterface()

```
virtual agvc_interface::AgvcInterface::~AgvcInterface ()  [virtual]
```

该类的文档由以下文件生成:

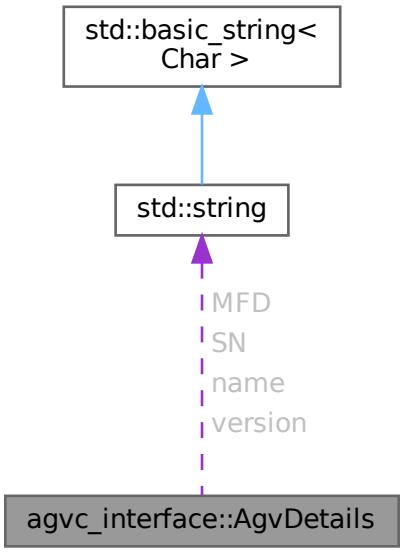
- include/agvc_interface/agvc_interface.h

12.4 agvc_interface::AgvDetails 结构体参考

agv 信息

`#include <type.h>`

agvc_interface::AgvDetails 的协作图:



Public 属性

- std::string name
  agv 的名称
- std::string version
  当前控制程序的版本号
- std::string MFD
  生产日期
- std::string SN
  S/N 码
- int communication_version
  bridge 版本信息
- int battery_version
  电池版本信息
- int loop_times
  电池循环次数
- float surplus_capacity
  电池剩余容量
- int left_motor_firmware_version
  左电机固件版本
- int right_motor_firmware_version
  右电机固件版本
- int master_board_firmware_version
  接口板固件版本信息
- int hardware_abstraction_version
  硬件抽象层版本信息
- int error_code_version
  错误码版本
- float max_speed
  agv 允许运行的最大线速度
- float max_angular
  agv 允许运行的最大角速度

12.4.1 详细描述

agv 信息
在文件 type.h 第 438 行定义.

12.4.2 类成员变量说明

battery_version

```
int agvc_interface::AgvDetails::battery_version
```

电池版本信息
在文件 type.h 第 445 行定义.

communication_version

```
int agvc_interface::AgvDetails::communication_version
```

bridge 版本信息
在文件 type.h 第 444 行定义.

error_code_version

```
int agvc_interface::AgvDetails::error_code_version
```

错误码版本
在文件 type.h 第 452 行定义.

hardware_abstraction_version

```
int agvc_interface::AgvDetails::hardware_abstraction_version
```

硬件抽象层版本信息
在文件 type.h 第 451 行定义.

left_motor_firmware_version

```
int agvc_interface::AgvDetails::left_motor_firmware_version
```

左电机固件版本
在文件 type.h 第 448 行定义.

loop_times

```
int agvc_interface::AgvDetails::loop_times
```

电池循环次数
在文件 type.h 第 446 行定义.

master_board_firmware_version

```
int agvc_interface::AgvDetails::master_board_firmware_version
```

接口板固件版本信息
在文件 type.h 第 450 行定义.

max_angular

```
float agvc_interface::AgvDetails::max_angular
```

agv 允许运行的最大角速度
在文件 type.h 第 454 行定义.

max_speed

```
float agvc_interface::AgvDetails::max_speed
```

agv 允许运行的最大线速度
在文件 type.h 第 453 行定义.

MFD

```
std::string agvc_interface::AgvDetails::MFD
```

生产日期
在文件 type.h 第 442 行定义.

name

```
std::string agvc_interface::AgvDetails::name
```

agv 的名称
在文件 type.h 第 440 行定义.

right_motor_firmware_version

```
int agvc_interface::AgvDetails::right_motor_firmware_version
```

右电机固件版本
在文件 type.h 第 449 行定义.

SN

```
std::string agvc_interface::AgvDetails::SN
```

S/N 码
在文件 type.h 第 443 行定义.

surplus_capacity

```
float agvc_interface::AgvDetails::surplus_capacity
```

电池剩余容量
在文件 type.h 第 447 行定义.

version

```
std::string agvc_interface::AgvDetails::version
```

当前控制程序的版本号
在文件 type.h 第 441 行定义.
该结构体的文档由以下文件生成:

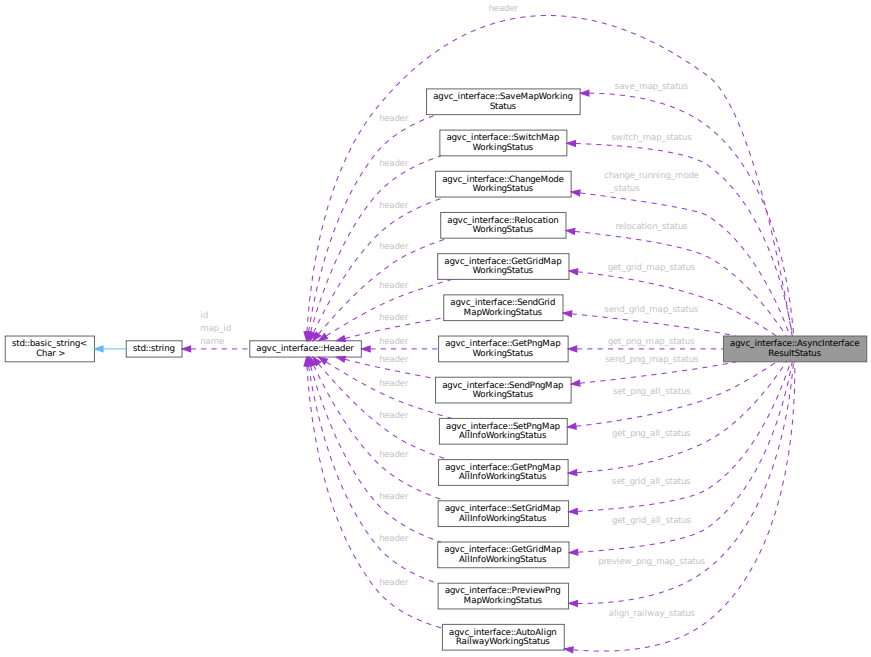
- include/agvc_interface/type.h

12.5 agvc_interface::AsyncInterfaceResultStatus 结构体参考

异步接口运行状态暂时存在 14 个异步接口：saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMap↩FromAgv getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv, autoAlignRailway

```
#include <type.h>
```

agvc_interface::AsyncInterfaceResultStatus 的协作图:



Public 属性

- Header header

  agv 的*SN* 码，*agv* 的名称，当前时间，地图的 *id*
- SaveMapWorkingStatus save_map_status

  保存地图 *(saveMap)* 接口的执行状态
- SwitchMapWorkingStatus switch_map_status

  切换地图 *(switchMap)* 接口的执行状态
- ChangeModeWorkingStatus change_running_mode_status

切换运行模式 *(changeRunningMode)* 接口的执行状态

- RelocationWorkingStatus relocation_status
  重定位 *(relocation)* 接口的执行状态
- GetGridMapWorkingStatus get_grid_map_status
  获取栅格地图 *(getGridMapFromAgv)* 接口的执行状态
- SendGridMapWorkingStatus send_grid_map_status
  发送栅格地图 *(sendGridMapToAgv)* 接口的执行状态
- GetPngMapWorkingStatus get_png_map_status
  获取 *png* 地图 *(getBase64PngMapFromAgv)* 接口的执行状态
- SendPngMapWorkingStatus send_png_map_status
  发送 *png* 地图 *(sendBase64PngMapToAgv)* 接口的执行状态
- SetPngMapAllInfoWorkingStatus set_png_all_status
  设置 *png* 地图全部信息 *(setPngMapAllInfo)* 接口的执行状态
- GetPngMapAllInfoWorkingStatus get_png_all_status
  获取 *png* 地图全部信息 *(getPngMapAllInfo)* 接口的执行状态
- SetGridMapAllInfoWorkingStatus set_grid_all_status
  设置栅格地图全部信息 *(setGridMapAllInfo)* 接口的执行状态
- GetGridMapAllInfoWorkingStatus get_grid_all_status
  获取栅格地图全部信息 *(getGridMapAllInfo)* 接口的执行状态
- PreviewPngMapWorkingStatus preview_png_map_status
  预览 *png* 地图 *(previewPngMapFromAgv)* 接口的执行状态
- AutoAlignRailwayWorkingStatus align_railway_status
  自动上轨 *(autoAlignRailway)* 接口的执行状态

12.5.1 详细描述

异步接口运行状态暂时存在 14 个异步接口：saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMap↩FromAgv getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv, autoAlignRailway

在文件 type.h 第 619 行定义.

12.5.2 类成员变量说明

align_railway_status

`AutoAlignRailwayWorkingStatus agvc_interface::AsyncInterfaceResultStatus::align_railway_status`
自动上轨 (autoAlignRailway) 接口的执行状态
在文件 type.h 第 635 行定义.

change_running_mode_status

`ChangeModeWorkingStatus agvc_interface::AsyncInterfaceResultStatus::change_running_mode_status`
切换运行模式 (changeRunningMode) 接口的执行状态
在文件 type.h 第 624 行定义.

get_grid_all_status

`GetGridMapAllInfoWorkingStatus agvc_interface::AsyncInterfaceResultStatus::get_grid_all_status`
获取栅格地图全部信息 (getGridMapAllInfo) 接口的执行状态
在文件 type.h 第 633 行定义.

get_grid_map_status

`GetGridMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::get_grid_map_status`
获取栅格地图 (getGridMapFromAgv) 接口的执行状态
在文件 type.h 第 626 行定义.

get_png_all_status

GetPngMapAllInfoWorkingStatus agvc_interface::AsyncInterfaceResultStatus::get_png_all_status
获取 png 地图全部信息 (getPngMapAllInfo) 接口的执行状态
在文件 type.h 第 631 行定义.

get_png_map_status

GetPngMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::get_png_map_status
获取 png 地图 (getBase64PngMapFromAgv) 接口的执行状态
在文件 type.h 第 628 行定义.

header

Header agvc_interface::AsyncInterfaceResultStatus::header
agv 的SN 码，agv 的名称，当前时间，地图的 id
在文件 type.h 第 621 行定义.

preview_png_map_status

PreviewPngMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::preview_png_map_status
预览 png 地图 (previewPngMapFromAgv) 接口的执行状态
在文件 type.h 第 634 行定义.

relocation_status

RelocationWorkingStatus agvc_interface::AsyncInterfaceResultStatus::relocation_status
重定位 (relocation) 接口的执行状态
在文件 type.h 第 625 行定义.

save_map_status

SaveMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::save_map_status
保存地图 (saveMap) 接口的执行状态
在文件 type.h 第 622 行定义.

send_grid_map_status

SendGridMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::send_grid_map_status
发送栅格地图 (sendGridMapToAgv) 接口的执行状态
在文件 type.h 第 627 行定义.

send_png_map_status

SendPngMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::send_png_map_status
发送 png 地图 (sendBase64PngMapToAgv) 接口的执行状态
在文件 type.h 第 629 行定义.

set_grid_all_status

SetGridMapAllInfoWorkingStatus agvc_interface::AsyncInterfaceResultStatus::set_grid_all_status
设置栅格地图全部信息 (setGridMapAllInfo) 接口的执行状态
在文件 type.h 第 632 行定义.

set_png_all_status

SetPngMapAllInfoWorkingStatus agvc_interface::AsyncInterfaceResultStatus::set_png_all_status
设置 png 地图全部信息 (setPngMapAllInfo) 接口的执行状态
在文件 type.h 第 630 行定义.

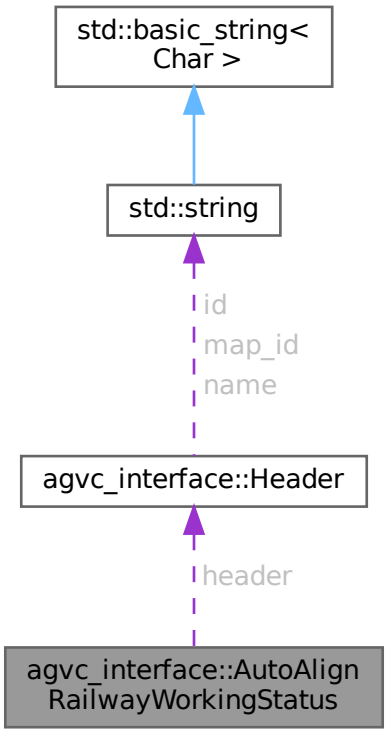
switch_map_status

`SwitchMapWorkingStatus agvc_interface::AsyncInterfaceResultStatus::switch_map_status`
切换地图 (switchMap) 接口的执行状态
在文件 type.h 第 623 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.6 agvc_interface::AutoAlignRailwayWorkingStatus 结构体参考

接口 autoAlignRailway 的工作状态
`#include <type.h>`
agvc_interface::AutoAlignRailwayWorkingStatus 的协作图:



Public 属性

- Header header
  header.id 代表下发的 *autoAlignRailway* 指令 *id*
- AutoAlignRailwayStatus status
  接口 *autoAlignRailway* 的运行状态 *NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.6.1 详细描述

接口 autoAlignRailway 的工作状态
在文件 type.h 第 606 行定义.

12.6.2 类成员变量说明

header

`Header agvc_interface::AutoAlignRailwayWorkingStatus::header`
header.id 代表下发的 autoAlignRailway 指令 id
在文件 type.h 第 608 行定义.

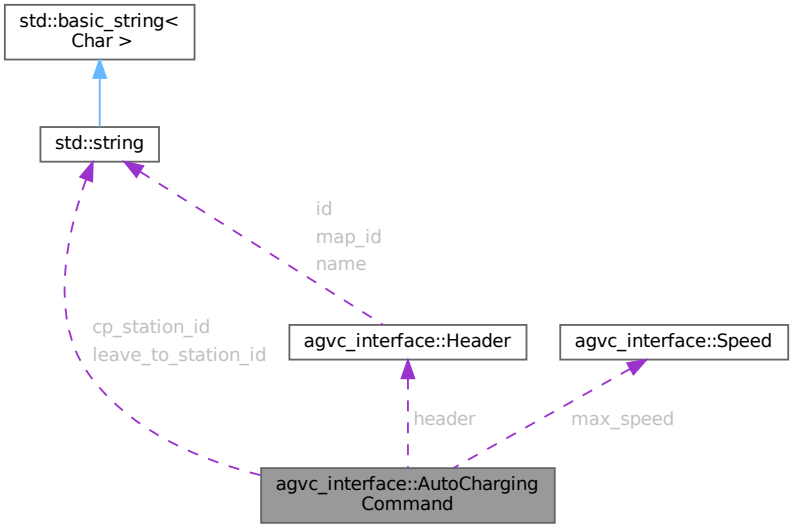
status

`AutoAlignRailwayStatus agvc_interface::AutoAlignRailwayWorkingStatus::status`
接口 autoAlignRailway 的运行状态 NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 609 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.7 agvc_interface::AutoChargingCommand 结构体参考

自动充电信息
`#include <type.h>`
agvc_interface::AutoChargingCommand 的协作图:



Public 属性

- Header header
  任务 *id*，任务名称，当前时间，地图 *id*
- AutoChargingType auto_charging_type
  自动充电类型
- std::string cp_station_id
  站点 *id*，上桩任务中，表示去往哪个充电站点进行充电；为空时前往最近充电站点
- std::string leave_to_station_id
  站点 *id*，下桩任务中，表示下桩导航到哪个站点；为空时导航到充电站点
- Speed max_speed
  充电过程中的最大行驶速度

12.7.1 详细描述

自动充电信息

自从

0.7.0

在文件 type.h 第 800 行定义.

12.7.2 类成员变量说明

auto_charging_type

```
AutoChargingType agvc_interface::AutoChargingCommand::auto_charging_type
```

自动充电类型
在文件 type.h 第 803 行定义.

cp_station_id

```
std::string agvc_interface::AutoChargingCommand::cp_station_id
```

站点 id，上桩任务中，表示去往哪个充电站点进行充电；为空时前往最近充电站点
在文件 type.h 第 804 行定义.

header

```
Header agvc_interface::AutoChargingCommand::header
```

任务 id，任务名称，当前时间，地图 id
在文件 type.h 第 802 行定义.

leave_to_station_id

```
std::string agvc_interface::AutoChargingCommand::leave_to_station_id
```

站点 id，下桩任务中，表示下桩导航到哪个站点；为空时导航到充电站点
在文件 type.h 第 805 行定义.

max_speed

```
Speed agvc_interface::AutoChargingCommand::max_speed
```

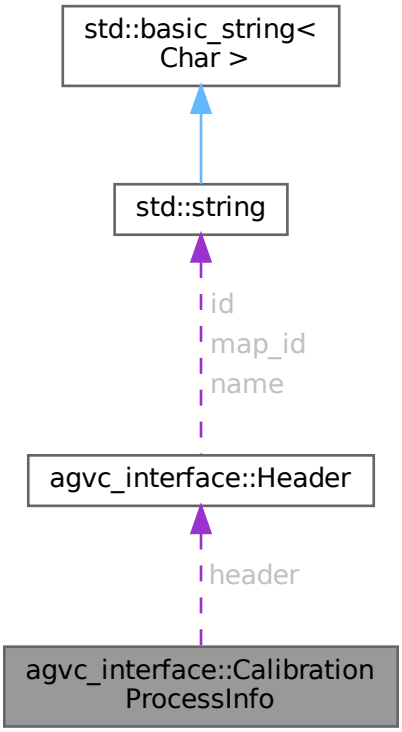
充电过程中的最大行驶速度
在文件 type.h 第 806 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.8 agvc_interface::CalibrationProcessInfo 结构体参考

```
#include <type.h>
```

agvc_interface::CalibrationProcessInfo 的协作图:



Public 属性

- Header header

  任务 *id*，任务名称，当前时间，地图 *id*
- CalibrationType type

  标定类型
- CalibrationStatus status

  标定状态
- uint32_t collected_data_count

  采集到多少组数据

12.8.1 详细描述

在文件 type.h 第 830 行定义.

12.8.2 类成员变量说明

collected_data_count

```
uint32_t agvc_interface::CalibrationProcessInfo::collected_data_count
```

采集到多少组数据

在文件 type.h 第 835 行定义.

header

`Header agvc_interface::CalibrationProcessInfo::header`
任务 id，任务名称，当前时间，地图 id
在文件 type.h 第 832 行定义.

status

`CalibrationStatus agvc_interface::CalibrationProcessInfo::status`
标定状态
在文件 type.h 第 834 行定义.

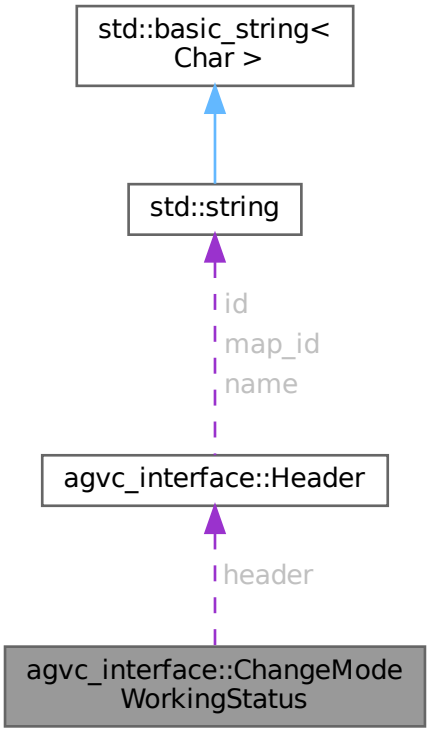
type

`CalibrationType agvc_interface::CalibrationProcessInfo::type`
标定类型
在文件 type.h 第 833 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.9 agvc_interface::ChangeModeWorkingStatus 结构体参考

接口 changeRunningMode 的工作状态
`#include <type.h>`
agvc_interface::ChangeModeWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *changeRunningMode* 指令 *id*
- ChangeRunningModeStatus status

  接口 *changeRunningMode* 的运行状态 *NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.9.1 详细描述

接口 changeRunningMode 的工作状态
在文件 type.h 第 506 行定义.

12.9.2 类成员变量说明

header

`Header agvc_interface::ChangeModeWorkingStatus::header`
header.id 代表下发的 changeRunningMode 指令 id
在文件 type.h 第 508 行定义.

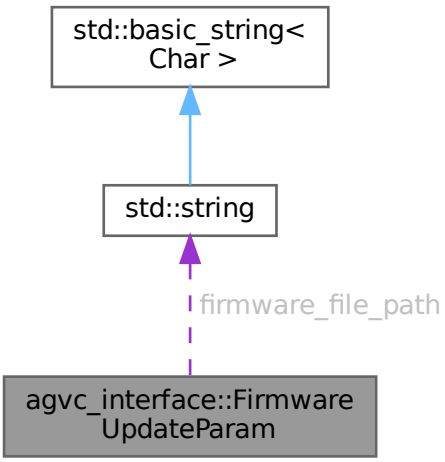
status

`ChangeRunningModeStatus agvc_interface::ChangeModeWorkingStatus::status`
接口 changeRunningMode 的运行状态 NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 509 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.10 agvc_interface::FirmwareUpdateParam 结构体参考

配置要更新固件的模块及固件路径
`#include <type.h>`
agvc_interface::FirmwareUpdateParam 的协作图:



Public 属性

- FirmwareUpdateMode left_motor

  左电机固件更新信息配置
- FirmwareUpdateMode right_motor

  右电机固件更新信息配置
- FirmwareUpdateMode master_board

  接口板固件更新信息配置
- std::string firmware_file_path

  代表固件包的路径例如 *"/root/update.firm"*

12.10.1 详细描述

配置要更新固件的模块及固件路径
在文件 type.h 第 812 行定义.

12.10.2 类成员变量说明

firmware_file_path

```
std::string agvc_interface::FirmwareUpdateParam::firmware_file_path
```

代表固件包的路径例如"/root/update.firm"
在文件 type.h 第 817 行定义.

left_motor

```
FirmwareUpdateMode agvc_interface::FirmwareUpdateParam::left_motor
```

左电机固件更新信息配置
在文件 type.h 第 814 行定义.

master_board

```
FirmwareUpdateMode agvc_interface::FirmwareUpdateParam::master_board
```

接口板固件更新信息配置
在文件 type.h 第 816 行定义.

right_motor

```
FirmwareUpdateMode agvc_interface::FirmwareUpdateParam::right_motor
```

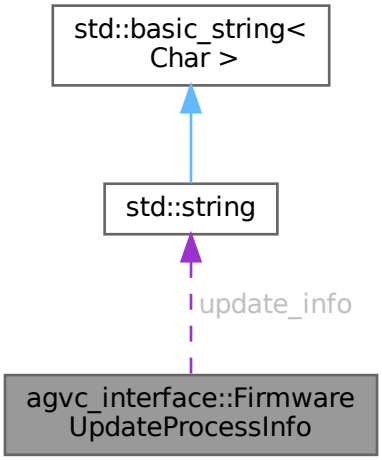
右电机固件更新信息配置
在文件 type.h 第 815 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.11 agvc_interface::FirmwareUpdateProcessInfo 结构体参考

固件更新过程信息
`#include <type.h>`

agvc_interface::FirmwareUpdateProcessInfo 的协作图:



Public 属性

- std::string update_info
  当前固件更新的步骤信息，*failed* 为升级失败, *none* 为无固件更新任务
- double update_progress
  固件更新进度信息，*1.0* 代表升级成功

12.11.1 详细描述

固件更新过程信息
在文件 type.h 第 823 行定义.

12.11.2 类成员变量说明

update_info

```
std::string agvc_interface::FirmwareUpdateProcessInfo::update_info
```

当前固件更新的步骤信息，failed 为升级失败, none 为无固件更新任务
在文件 type.h 第 825 行定义.

update_progress

```
double agvc_interface::FirmwareUpdateProcessInfo::update_progress
```

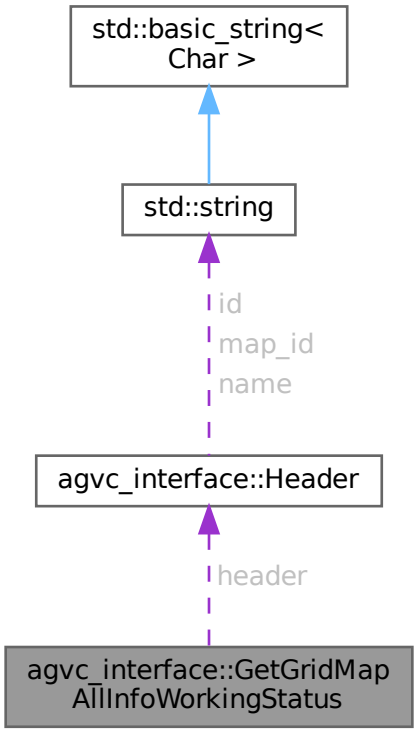
固件更新进度信息，1.0 代表升级成功
在文件 type.h 第 827 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.12 agvc_interface::GetGridMapAllInfoWorkingStatus 结构体参考

接口 getGridMapAllInfo 的工作状态
`#include <type.h>`

agvc_interface::GetGridMapAllInfoWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *getGridMapAllInfo* 指令 *id*
- GetGridMapAllInfoStatus status

  接口 *getGridMapAllInfo* 的运行状态 *NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.12.1 详细描述

接口 getGridMapAllInfo 的工作状态
在文件 type.h 第 586 行定义.

12.12.2 类成员变量说明

header

```
Header agvc_interface::GetGridMapAllInfoWorkingStatus::header
```

header.id 代表下发的 getGridMapAllInfo 指令 id
在文件 type.h 第 588 行定义.

status

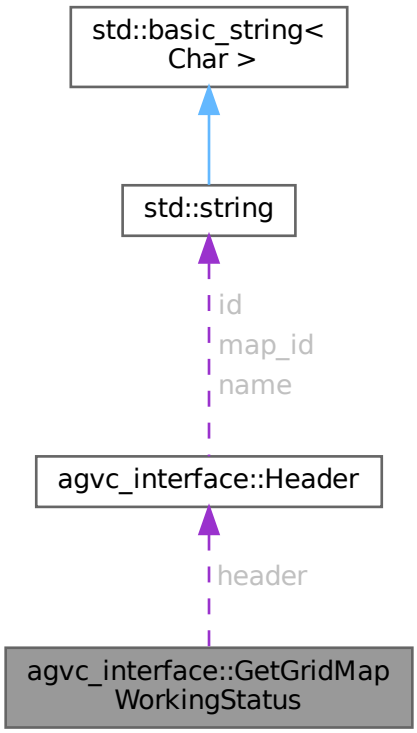
```
GetGridMapAllInfoStatus agvc_interface::GetGridMapAllInfoWorkingStatus::status
```

接口 getGridMapAllInfo 的运行状态 NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 589 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.13 agvc_interface::GetGridMapWorkingStatus 结构体参考

接口 getGridMapFromAgv 的工作状态
`#include <type.h>`
agvc_interface::GetGridMapWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *getGridMapFromAgv* 指令 *id*
- GetGridMapStatus status

  接口 *getGridMapFromAgv* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.13.1 详细描述

接口 getGridMapFromAgv 的工作状态
在文件 type.h 第 543 行定义.

12.13.2 类成员变量说明

header

```
Header agvc_interface::GetGridMapWorkingStatus::header
```

header.id 代表下发的 getGridMapFromAgv 指令 id
在文件 type.h 第 545 行定义.

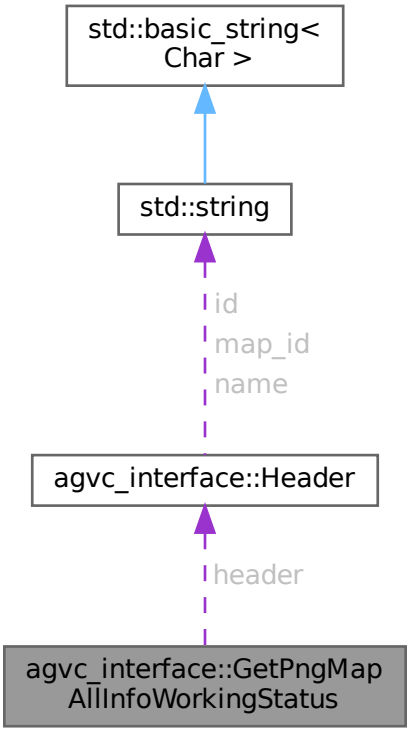
status

GetGridMapStatus agvc_interface::GetGridMapWorkingStatus::status
接口 getGridMapFromAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 546 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.14 agvc_interface::GetPngMapAllInfoWorkingStatus 结构体参考

接口 getPngMapAllInfo 的工作状态
`#include <type.h>`
agvc_interface::GetPngMapAllInfoWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *getPngMapAllInfo* 指令 *id*
- GetPngMapAllInfoStatus status

  接口 *getPngMapAllInfo* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.14.1 详细描述

接口 getPngMapAllInfo 的工作状态
在文件 type.h 第 534 行定义.

12.14.2 类成员变量说明

header

`Header agvc_interface::GetPngMapAllInfoWorkingStatus::header`
header.id 代表下发的 getPngMapAllInfo 指令 id
在文件 type.h 第 536 行定义.

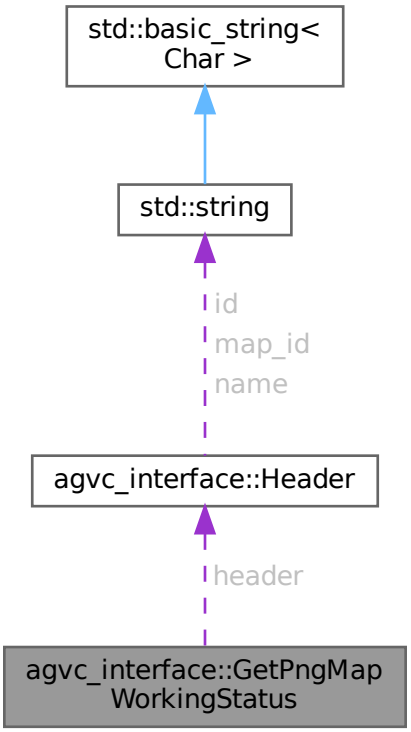
status

`GetPngMapAllInfoStatus agvc_interface::GetPngMapAllInfoWorkingStatus::status`
接口 getPngMapAllInfo 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 537 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.15 agvc_interface::GetPngMapWorkingStatus 结构体参考

`#include <type.h>`
agvc_interface::GetPngMapWorkingStatus 的协作图:



Public 属性

- Header header
  header.id 代表下发的 *getBase64PngMapFromAgv* 指令 *id*
- GetPngMapStatus status
  接口 *getBase64PngMapFromAgv* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.15.1 详细描述

在文件 type.h 第 558 行定义.

12.15.2 类成员变量说明

header

`Header agvc_interface::GetPngMapWorkingStatus::header`
header.id 代表下发的 getBase64PngMapFromAgv 指令 id
在文件 type.h 第 560 行定义.

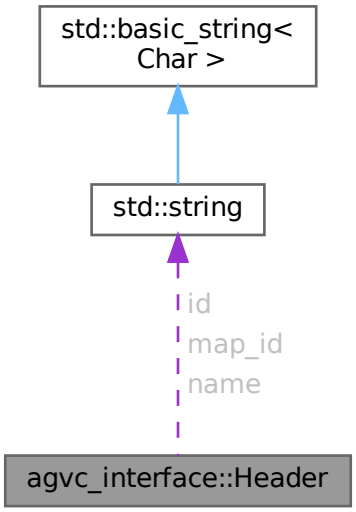
status

`GetPngMapStatus agvc_interface::GetPngMapWorkingStatus::status`
接口 getBase64PngMapFromAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 561 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.16 agvc_interface::Header 结构体参考

头部信息
`#include <type.h>`
agvc_interface::Header 的协作图:



Public 属性

- std::string id
  uuid
- std::string name
  名称
- int64_t stamp

时间戳，*UTC* 时间 *(1970* 年 *01* 月 *01* 日 *00* 时 *00* 分 *00* 秒 ~ 至今*)*，单位：纳秒

- std::string map_id
  当前地图的 *id*, 形式为 *uuid*：例如 *a5e7c6de-4463-443e-8b25-ccbfd6a27aee*, 与地图文件名字相同
- int error_code = -1
  错误码

12.16.1 详细描述

头部信息
在文件 type.h 第 425 行定义.

12.16.2 类成员变量说明

error_code

```
int agvc_interface::Header::error_code = -1
```

错误码
在文件 type.h 第 432 行定义.

id

```
std::string agvc_interface::Header::id
```

uuid
在文件 type.h 第 427 行定义.

map_id

```
std::string agvc_interface::Header::map_id
```

当前地图的 id, 形式为 uuid：例如 a5e7c6de-4463-443e-8b25-ccbfd6a27aee, 与地图文件名字相同
在文件 type.h 第 430 行定义.

name

```
std::string agvc_interface::Header::name
```

名称
在文件 type.h 第 428 行定义.

stamp

```
int64_t agvc_interface::Header::stamp
```

时间戳，UTC 时间 (1970 年 01 月 01 日 00 时 00 分 00 秒 ~ 至今)，单位：纳秒
在文件 type.h 第 429 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.17 agvc_interface::InputParser 类参考

解析输入
`#include <rtde.h>`

Public 成员函数

- InputParser ()
- ~InputParser ()
- bool popBool ()
- int popInt32 ()
- int64_t popInt64 ()
- int16_t popInt16 ()

- double popDouble ()
- char popChar ()
- std::string popString ()
- std::vector< int > popVectorInt ()
- std::vector< int16_t > popVectorInt16 ()
- std::vector< double > popVectorDouble ()
- std::vector< std::vector< double > > popVectorVectorDouble ()
- agvc_interface::Pose2d popPose2d ()
- agvc_interface::RunningInfo popRunningInfo ()
- agvc_interface::AgvDetails popAgvDetails ()
- agvc_interface::NavInfo popNavInfo ()
- std::vector< agvc_interface::Point2d > popVectorPoint2d ()
- std::vector< agvc_interface::Header > popVectorHeader ()

Private 属性

- friend RtdeClient
- Impl * impl

12.17.1 详细描述

解析输入
在文件 rtde.h 第 48 行定义.

12.17.2 构造及析构函数说明

InputParser()

```
agvc_interface::InputParser::InputParser ()
```

~InputParser()

```
agvc_interface::InputParser::~InputParser ()
```

12.17.3 成员函数说明

popAgvDetails()

```
agvc_interface::AgvDetails agvc_interface::InputParser::popAgvDetails ()
```

popBool()

```
bool agvc_interface::InputParser::popBool ()
```

popChar()

```
char agvc_interface::InputParser::popChar ()
```

popDouble()

```
double agvc_interface::InputParser::popDouble ()
```

popInt16()

```
int16_t agvc_interface::InputParser::popInt16 ()
```

popInt32()

```
int agvc_interface::InputParser::popInt32 ()
```

popInt64()

```
int64_t agvc_interface::InputParser::popInt64 ()
```

popNavInfo()

```
agvc_interface::NavInfo agvc_interface::InputParser::popNavInfo ()
```

popPose2d()

```
agvc_interface::Pose2d agvc_interface::InputParser::popPose2d ()
```

popRunningInfo()

```
agvc_interface::RunningInfo agvc_interface::InputParser::popRunningInfo ()
```

popString()

```
std::string agvc_interface::InputParser::popString ()
```

popVectorDouble()

```
std::vector< double > agvc_interface::InputParser::popVectorDouble ()
```

popVectorHeader()

```
std::vector< agvc_interface::Header > agvc_interface::InputParser::popVectorHeader ()
```

popVectorInt()

```
std::vector< int > agvc_interface::InputParser::popVectorInt ()
```

popVectorInt16()

```
std::vector< int16_t > agvc_interface::InputParser::popVectorInt16 ()
```

popVectorPoint2d()

```
std::vector< agvc_interface::Point2d > agvc_interface::InputParser::popVectorPoint2d ()
```

popVectorVectorDouble()

```
std::vector< std::vector< double > > agvc_interface::InputParser::popVectorVectorDouble ()
```

12.17.4 类成员变量说明

impl

```
Impl* agvc_interface::InputParser::impl  [private]
```

在文件 rtde.h 第 75 行定义.

RtdeClient

```
friend agvc_interface::InputParser::RtdeClient  [private]
```

在文件 rtde.h 第 73 行定义.
该类的文档由以下文件生成:

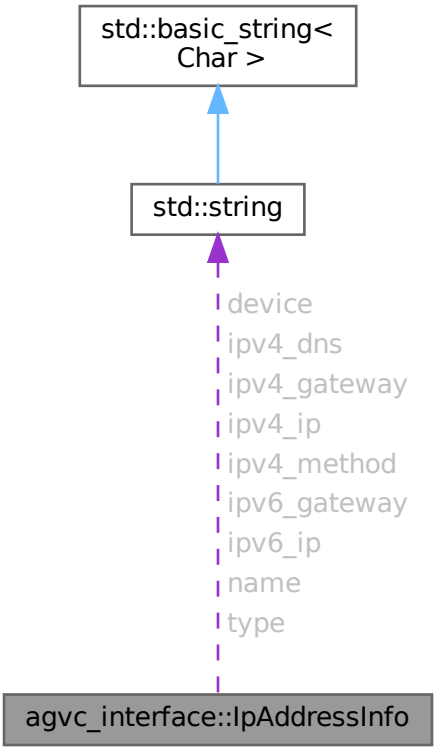
- include/agvc_interface/rtde.h

12.18 agvc_interface::IpAddressInfo 结构体参考

IP 地址

`#include <type.h>`

agvc_interface::IpAddressInfo 的协作图:



Public 属性

- std::string name
  连接名称
- std::string type
  类型
- std::string device
  网卡名称
- std::string ipv4_ip
  IPv4 地址及*CIDR* 码，为空时无效
- std::string ipv4_gateway
  网关
- std::string ipv4_method
  自动分配或者手动连接，对应字段 *auto, manual*
- std::string ipv4_dns
  第一个*DNS*
- std::string ipv6_ip
  IPv6 地址及*CIDR* 码，为空时无效
- std::string ipv6_gateway

12.18.1 详细描述

IP 地址
在文件 type.h 第 737 行定义.

12.18.2 类成员变量说明

device

```
std::string agvc_interface::IpAddressInfo::device
```

网卡名称
在文件 type.h 第 741 行定义.

ipv4_dns

```
std::string agvc_interface::IpAddressInfo::ipv4_dns
```

第一个DNS
在文件 type.h 第 745 行定义.

ipv4_gateway

```
std::string agvc_interface::IpAddressInfo::ipv4_gateway
```

网关
在文件 type.h 第 743 行定义.

ipv4_ip

```
std::string agvc_interface::IpAddressInfo::ipv4_ip
```

IPv4 地址及CIDR 码，为空时无效
在文件 type.h 第 742 行定义.

ipv4_method

```
std::string agvc_interface::IpAddressInfo::ipv4_method
```

自动分配或者手动连接，对应字段 auto, manual
在文件 type.h 第 744 行定义.

ipv6_gateway

```
std::string agvc_interface::IpAddressInfo::ipv6_gateway
```

在文件 type.h 第 747 行定义.

ipv6_ip

```
std::string agvc_interface::IpAddressInfo::ipv6_ip
```

IPv6 地址及CIDR 码，为空时无效
在文件 type.h 第 746 行定义.

name

```
std::string agvc_interface::IpAddressInfo::name
```

连接名称
在文件 type.h 第 739 行定义.

type

```
std::string agvc_interface::IpAddressInfo::type
```

类型
在文件 type.h 第 740 行定义.
该结构体的文档由以下文件生成:

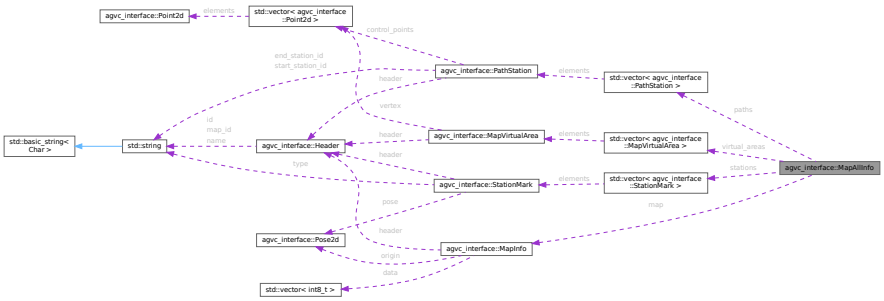
- include/agvc_interface/type.h

12.19 agvc_interface::MapAllInfo 结构体参考

包含当前地图、当前地图上的站点、当前地图上的路径、当前地图上的虚拟区域信息

`#include <type.h>`

agvc_interface::MapAllInfo 的协作图:



Public 属性

- MapInfo map

  当前地图信息

- std::vector< StationMark > stations

  当前地图上的站点信息

- std::vector< PathStation > paths

  当前地图上的路径信息

- std::vector< MapVirtualArea > virtual_areas

  当前地图上的虚拟区域信息

12.19.1 详细描述

包含当前地图、当前地图上的站点、当前地图上的路径、当前地图上的虚拟区域信息

在文件 type.h 第 726 行定义.

12.19.2 类成员变量说明

map

```
MapInfo agvc_interface::MapAllInfo::map
```

当前地图信息

在文件 type.h 第 728 行定义.

paths

```
std::vector<PathStation> agvc_interface::MapAllInfo::paths
```

当前地图上的路径信息

在文件 type.h 第 730 行定义.

stations

```
std::vector<StationMark> agvc_interface::MapAllInfo::stations
```

当前地图上的站点信息

在文件 type.h 第 729 行定义.

virtual_areas

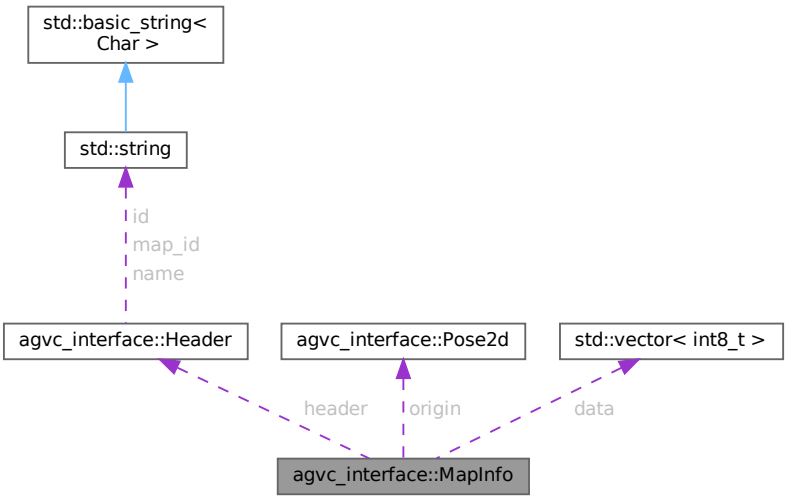
```
std::vector<MapVirtualArea> agvc_interface::MapAllInfo::virtual_areas
```

当前地图上的虚拟区域信息
在文件 type.h 第 731 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.20 agvc_interface::MapInfo 结构体参考

地图信息
`#include <type.h>`
agvc_interface::MapInfo 的协作图:



Public 属性

- Header header

  地图 *id*，地图名称，当前时间，*map_id* 为地图的唯一 *id*
- double resolution

  分辨率，每个正方形栅格/像素的宽度，单位 *m*
- uint32_t width

  地图宽度，多少个栅格/像素宽度
- uint32_t height

  地图高度，多少个栅格/像素高度
- Pose2d origin

  {x(m) y(m) theta(rad)} 地图原点在真实地图下的坐标，指地图原点对应的实际位置坐标。一张地图中，左下角为地图原点；
- MapFormat format

  地图格式
- std::vector< int8_t > data

  地图数据信息

12.20.1 详细描述

地图信息
在文件 type.h 第 665 行定义.

12.20.2 类成员变量说明

data

```
std::vector<int8_t> agvc_interface::MapInfo::data
```

地图数据信息
在文件 type.h 第 675 行定义.

format

```
MapFormat agvc_interface::MapInfo::format
```

地图格式
在文件 type.h 第 674 行定义.

header

```
Header agvc_interface::MapInfo::header
```

地图 id，地图名称，当前时间，map_id 为地图的唯一 id
在文件 type.h 第 667 行定义.

height

```
uint32_t agvc_interface::MapInfo::height
```

地图高度，多少个栅格/像素高度
在文件 type.h 第 670 行定义.

origin

```
Pose2d agvc_interface::MapInfo::origin
```

{x(m) y(m) theta(rad)} 地图原点在真实地图下的坐标，指地图原点对应的实际位置坐标。一张地图中，左下角为地图原点;
在文件 type.h 第 671 行定义.

resolution

```
double agvc_interface::MapInfo::resolution
```

分辨率，每个正方形栅格/像素的宽度，单位 m
在文件 type.h 第 668 行定义.

width

```
uint32_t agvc_interface::MapInfo::width
```

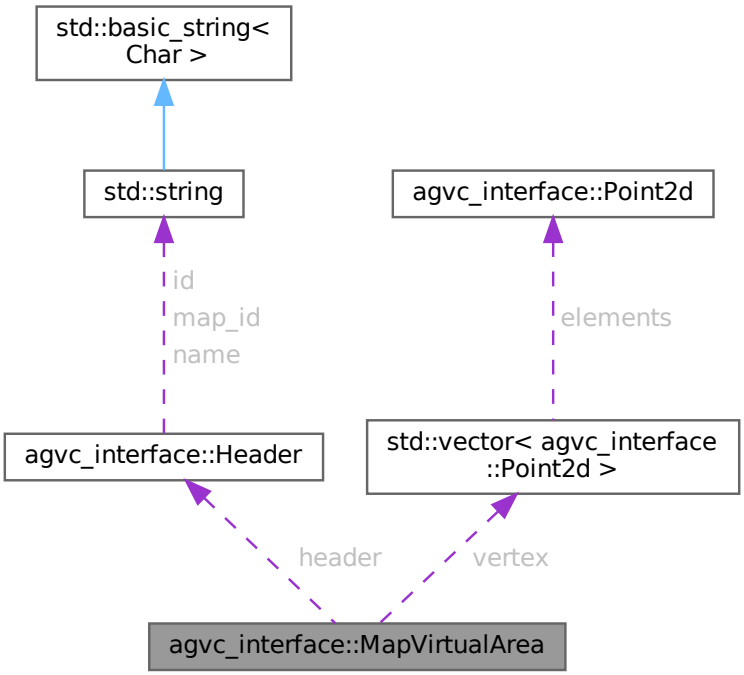
地图宽度，多少个栅格/像素宽度
在文件 type.h 第 669 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.21 agvc_interface::MapVirtualArea 结构体参考

地图中的虚拟区域
`#include <type.h>`

agvc_interface::MapVirtualArea 的协作图:



Public 属性

- Header header
  虚拟区域的 *id*，虚拟区域的名称，当前时间，虚拟区域所在地图的 *id*
- MapVirtualAreaType type
  虚拟区域的类型
- MapVirtualAreaShape shape
  虚拟区域形状
- double width_speed
  虚拟墙的宽度/膨胀半径/减速区域的最大速度
- std::vector< Point2d > vertex
  虚拟区域的顶点 *{{x(m) y(m)},{}, ...}* 直线、多边形：按照顺时针顺序把顶点加入到 *vertex* 中圆弧：圆弧起点、圆弧上一点、圆弧终点圆形：圆上按顺序排序的任意不共线三个以上的点 *(包含 3 个)*

12.21.1 详细描述

地图中的虚拟区域
在文件 type.h 第 711 行定义.

12.21.2 类成员变量说明

header

```
Header agvc_interface::MapVirtualArea::header
```

虚拟区域的 id，虚拟区域的名称，当前时间，虚拟区域所在地图的 id
在文件 type.h 第 713 行定义.

shape

MapVirtualAreaShape agvc_interface::MapVirtualArea::shape
虚拟区域形状
在文件 type.h 第 715 行定义.

type

MapVirtualAreaType agvc_interface::MapVirtualArea::type
虚拟区域的类型
在文件 type.h 第 714 行定义.

vertex

std::vector<Point2d> agvc_interface::MapVirtualArea::vertex
虚拟区域的顶点 {{x(m) y(m)},{}, ...} 直线、多边形：按照顺时针顺序把顶点加入到 vertex 中圆弧：圆弧起点、圆弧上一点、圆弧终点圆形：圆上按顺序排序的任意不共线三个以上的点 (包含 3 个)
在文件 type.h 第 717 行定义.

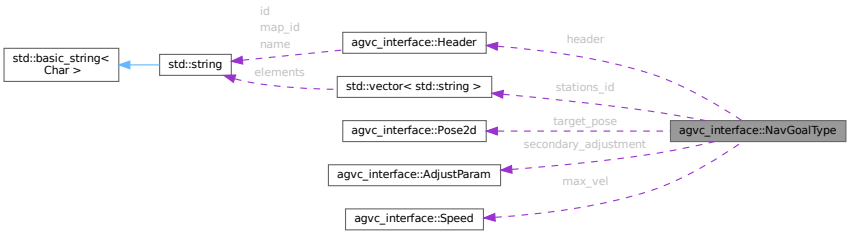
width_speed

double agvc_interface::MapVirtualArea::width_speed
虚拟墙的宽度/膨胀半径/减速区域的最大速度
在文件 type.h 第 716 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.22 agvc_interface::NavGoalType 结构体参考

使 agv 导航到目标位置（控制信息）
#include <type.h>
agvc_interface::NavGoalType 的协作图:



Public 属性

- Header header
  任务 *id*，任务名称，当前时间，地图 *id*
- NavType type
  导航类型
- Pose2d target_pose
  目标位姿 *{x(m) y(m) theta(rad)}* 导航类型为自由导航到任意点时有效，其他无效
- std::vector< std::string > stations_id
  导航经过的站点 *id*, 最后一个站点为导航目标位置导航类型为自由导航到站点时，或路径导航到站点时有效，其他无效
- bool forward_flag

agv 正向行驶或倒车行驶

- bool goal_end_rotate

  到达目标点后是否根据站点方向旋转，*true*：旋转 *false*：不旋转

- AdjustParam secondary_adjustment

  到达目标点后是否进行二次调整，贴合目标点位置，*true*：到点调整 *false*：到点不调整

- Speed max_vel

  最大速度 *v,w*

12.22.1 详细描述

使 agv 导航到目标位置（控制信息）
在文件 type.h 第 762 行定义.

12.22.2 类成员变量说明

forward_flag

```
bool agvc_interface::NavGoalType::forward_flag
```

agv 正向行驶或倒车行驶
在文件 type.h 第 770 行定义.

goal_end_rotate

```
bool agvc_interface::NavGoalType::goal_end_rotate
```

到达目标点后是否根据站点方向旋转，true：旋转 false：不旋转
在文件 type.h 第 771 行定义.

header

```
Header agvc_interface::NavGoalType::header
```

任务 id，任务名称，当前时间，地图 id
在文件 type.h 第 764 行定义.

max_vel

```
Speed agvc_interface::NavGoalType::max_vel
```

最大速度 v,w
在文件 type.h 第 775 行定义.

secondary_adjustment

```
AdjustParam agvc_interface::NavGoalType::secondary_adjustment
```

到达目标点后是否进行二次调整，贴合目标点位置, true：到点调整 false：到点不调整
在文件 type.h 第 773 行定义.

stations_id

```
std::vector<std::string> agvc_interface::NavGoalType::stations_id
```

导航经过的站点 id, 最后一个站点为导航目标位置导航类型为自由导航到站点时，或路径导航到站点时有效，其他无效
在文件 type.h 第 768 行定义.

target_pose

```
Pose2d agvc_interface::NavGoalType::target_pose
```

目标位姿 {x(m) y(m) theta(rad)} 导航类型为自由导航到任意点时有效，其他无效
在文件 type.h 第 766 行定义.

type

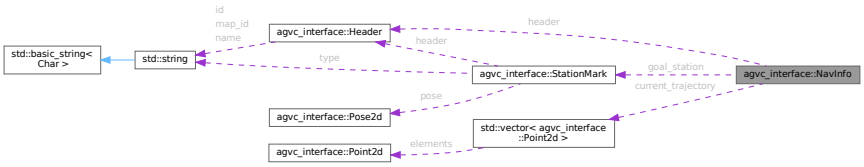
```
NavType agvc_interface::NavGoalType::type
```

导航类型
在文件 type.h 第 765 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.23 agvc_interface::NavInfo 结构体参考

导航状态信息 / 自动充电状态信息
`#include <type.h>`
agvc_interface::NavInfo 的协作图:



Public 属性

- Header header
  任务 *id*，任务名称，当前时间，地图 *id*
- NavStatus status
  导航状态
- NavType type
- NavType nav_type
  导航类型
- AutoChargingType auto_charging_type
  自动充电类型
- StationMark goal_station
  agv 行驶路径目标站点
- std::vector< Point2d > current_trajectory
  还没经过的路径点 *{{x(m) y(m)},{}, ...}*

12.23.1 详细描述

导航状态信息 / 自动充电状态信息
导航模式时，获取导航状态信息
自动充电模式时，获取自动充电状态信息，type 与 current_trajectory 无效
在文件 type.h 第 784 行定义.

12.23.2 类成员变量说明

auto_charging_type

```
AutoChargingType agvc_interface::NavInfo::auto_charging_type
```

自动充电类型
在文件 type.h 第 791 行定义.

current_trajectory

std::vector<Point2d> agvc_interface::NavInfo::current_trajectory
还没经过的路径点 {{x(m) y(m)},{}, ...}
在文件 type.h 第 793 行定义.

goal_station

StationMark agvc_interface::NavInfo::goal_station
agv 行驶路径目标站点
在文件 type.h 第 792 行定义.

header

Header agvc_interface::NavInfo::header
任务 id，任务名称，当前时间，地图 id
在文件 type.h 第 786 行定义.

nav_type

NavType agvc_interface::NavInfo::nav_type
导航类型
在文件 type.h 第 790 行定义.

status

NavStatus agvc_interface::NavInfo::status
导航状态
在文件 type.h 第 787 行定义.

type

NavType agvc_interface::NavInfo::type
在文件 type.h 第 789 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.24 agvc_interface::OutputBuilder 类参考

向输出数据中增加
`#include <rtde.h>`

Public 成员函数

- OutputBuilder ()
- ~OutputBuilder ()
- OutputBuilder & push (int val)
- OutputBuilder & push (double val)
- OutputBuilder & push (const std::vector< double > &val)
- OutputBuilder & push (const std::tuple< int, bool > &val)
- OutputBuilder & push (int16_t &val)
- OutputBuilder & push (const std::vector< int16_t > &val)
- OutputBuilder & push (const std::vector< int > &val)
- OutputBuilder & push (const std::string &val)
- OutputBuilder & push (char val)
- OutputBuilder & push (const agvc_interface::RtdeRecipe &val)

Private 属性

- friend RtdeClient
- Impl * impl

12.24.1 详细描述

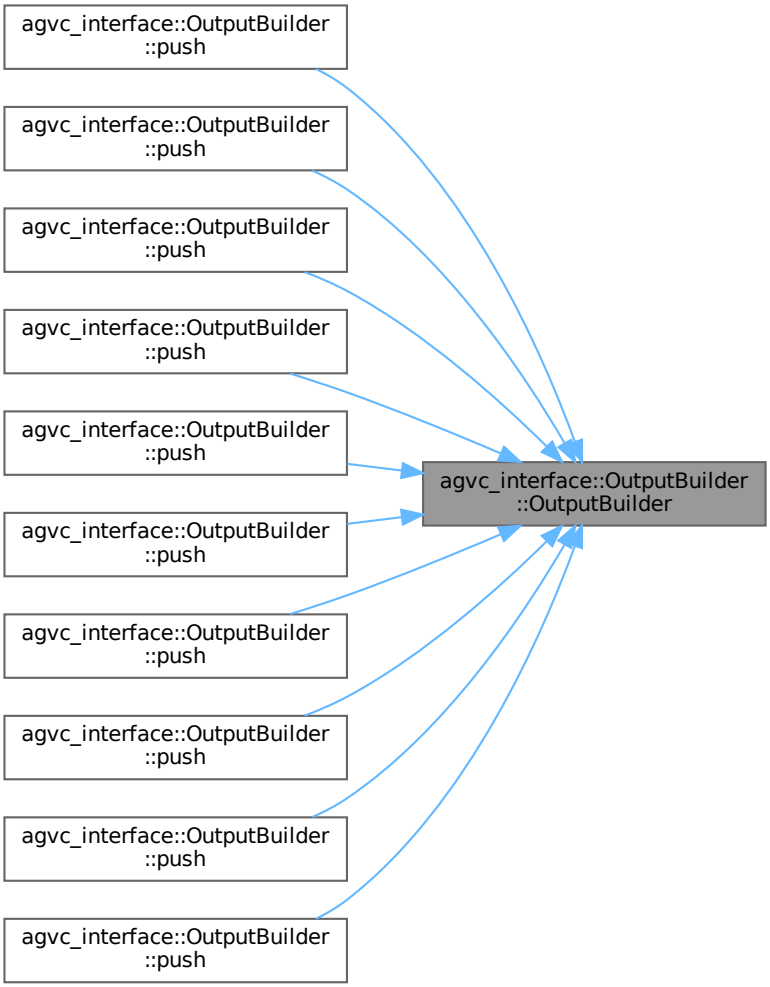
向输出数据中增加
在文件 rtde.h 第 23 行定义.

12.24.2 构造及析构函数说明

OutputBuilder()

```
agvc_interface::OutputBuilder::OutputBuilder ()
```

被这些函数引用 push(), push(), push(), push(), push(), push(), push(), push(), push() , 以及 push().
这是这个函数的调用关系图:



~OutputBuilder()

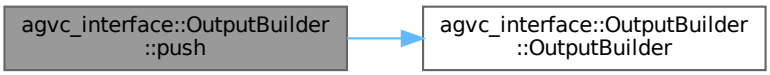
```
agvc_interface::OutputBuilder::~OutputBuilder ()
```

12.24.3 成员函数说明

push() [1/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        char val)
```

引用了 OutputBuilder().
函数调用图:



push() [2/10]

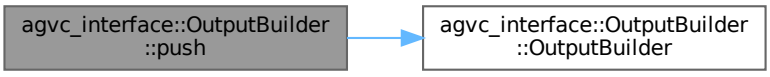
```
OutputBuilder & agvc_interface::OutputBuilder::push (
        const agvc_interface::RtdeRecipe & val)
```

引用了 OutputBuilder().
函数调用图:

push() [3/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        const std::string & val)
```

引用了 OutputBuilder().
函数调用图:



push() [4/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        const std::tuple< int, bool > & val)
```

引用了 OutputBuilder().

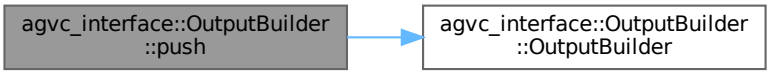
函数调用图:

push() [5/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
            const std::vector< double > & val)
```

引用了 OutputBuilder().

函数调用图:

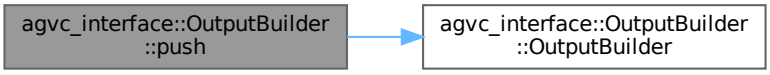


push() [6/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
            const std::vector< int > & val)
```

引用了 OutputBuilder().

函数调用图:



push() [7/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
            const std::vector< int16_t > & val)
```

引用了 OutputBuilder().

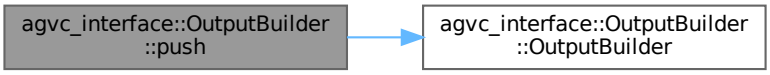
函数调用图:

push() [8/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        double val)
```

引用了 OutputBuilder().

函数调用图:

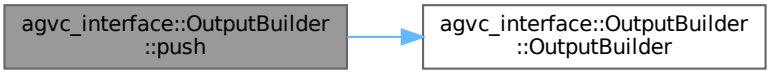


push() [9/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        int val)
```

引用了 OutputBuilder().

函数调用图:

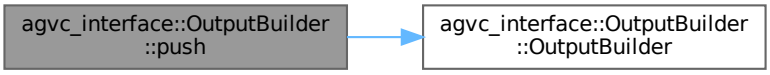


push() [10/10]

```
OutputBuilder & agvc_interface::OutputBuilder::push (
        int16_t & val)
```

引用了 OutputBuilder().

函数调用图:



12.24.4 类成员变量说明

impl

```
Impl* agvc_interface::OutputBuilder::impl  [private]
```

在文件 rtde.h 第 44 行定义.

RtdeClient

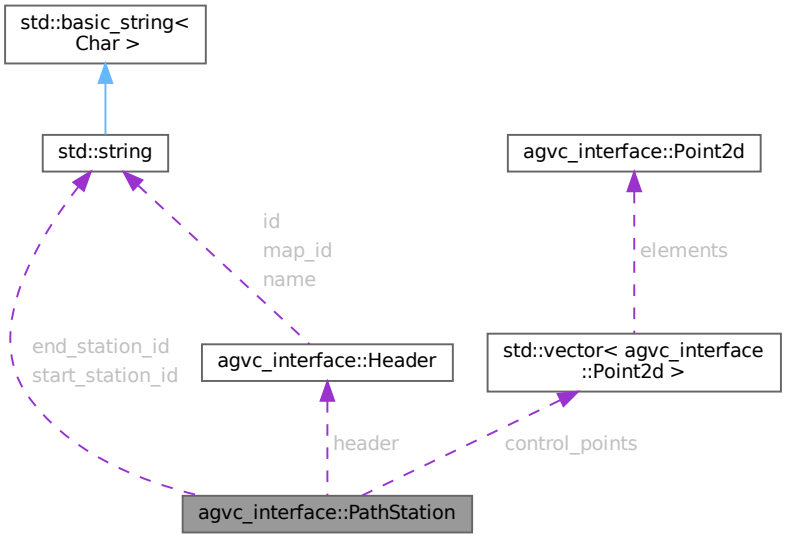
```
friend agvc_interface::OutputBuilder::RtdeClient  [private]
```

在文件 rtde.h 第 42 行定义.
该类的文档由以下文件生成:

- include/agvc_interface/rtde.h

12.25 agvc_interface::PathStation 结构体参考

通过站点表示路径
`#include <type.h>`
agvc_interface::PathStation 的协作图:



Public 属性

- Header header
  路径 *id*，路径名称，当前时间，路径所在地图的 *id*
- PathShape shape
  路径的形状
- bool use_direction
  生成站点时是否使用站点方向
- double max_speed
  agv 经过该路径时的最大速度
- std::string start_station_id
  路径起点站点 *id*
- std::string end_station_id
  路径终点站点 *id*
- std::vector< Point2d > control_points
  路径的控制点，*{{x,y},{}, ...}*

12.25.1 详细描述

通过站点表示路径
在文件 type.h 第 651 行定义.

12.25.2 类成员变量说明

control_points

```
std::vector<Point2d> agvc_interface::PathStation::control_points
```

路径的控制点，{{x,y},{}, ...}
在文件 type.h 第 659 行定义.

end_station_id

```
std::string agvc_interface::PathStation::end_station_id
```

路径终点站点 id
在文件 type.h 第 658 行定义.

header

```
Header agvc_interface::PathStation::header
```

路径 id，路径名称，当前时间，路径所在地图的 id
在文件 type.h 第 653 行定义.

max_speed

```
double agvc_interface::PathStation::max_speed
```

agv 经过该路径时的最大速度
在文件 type.h 第 656 行定义.

shape

```
PathShape agvc_interface::PathStation::shape
```

路径的形状
在文件 type.h 第 654 行定义.

start_station_id

```
std::string agvc_interface::PathStation::start_station_id
```

路径起点站点 id
在文件 type.h 第 657 行定义.

use_direction

bool agvc_interface::PathStation::use_direction
生成站点时是否使用站点方向
在文件 type.h 第 655 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.26 agvc_interface::Point2d 结构体参考

二维点
#include <type.h>

Public 属性

- float x
  单位 m
- float y
  单位 m

12.26.1 详细描述

二维点
在文件 type.h 第 397 行定义.

12.26.2 类成员变量说明

x

float agvc_interface::Point2d::x
单位 m
在文件 type.h 第 399 行定义.

y

float agvc_interface::Point2d::y
单位 m
在文件 type.h 第 400 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.27 agvc_interface::Pose2d 结构体参考

二维位姿
#include <type.h>

Public 属性

- double x
  单位 m
- double y
  单位 m
- double yaw
  单位 rad

12.27.1 详细描述

二维位姿

在文件 type.h 第 406 行定义.

12.27.2 类成员变量说明

x

```
double agvc_interface::Pose2d::x
```

单位 m

在文件 type.h 第 408 行定义.

y

```
double agvc_interface::Pose2d::y
```

单位 m

在文件 type.h 第 409 行定义.

yaw

```
double agvc_interface::Pose2d::yaw
```

单位 rad

在文件 type.h 第 410 行定义.

该结构体的文档由以下文件生成:

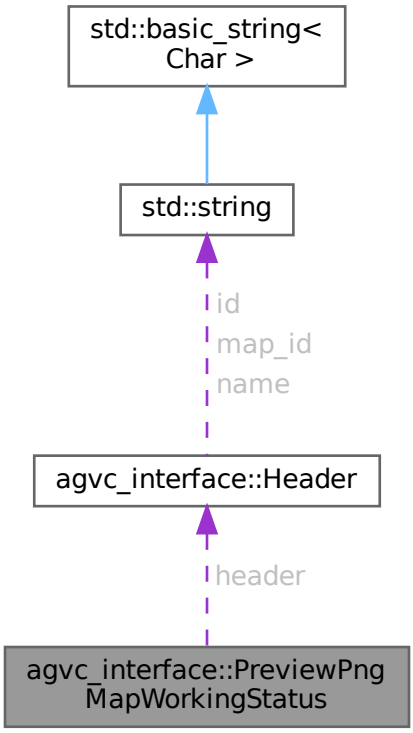
- include/agvc_interface/type.h

12.28 agvc_interface::PreviewPngMapWorkingStatus 结构体参考

接口 previewPngMapFromAgv 的工作状态

```
#include <type.h>
```

agvc_interface::PreviewPngMapWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *previewPngMapFromAgv* 指令 *id*
- PreviewPngMapStatus status

  接口 *previewPngMapFromAgv* 的运行状态 *NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.28.1 详细描述

接口 previewPngMapFromAgv 的工作状态
在文件 type.h 第 596 行定义.

12.28.2 类成员变量说明

header

```
Header agvc_interface::PreviewPngMapWorkingStatus::header
```

header.id 代表下发的 previewPngMapFromAgv 指令 id
在文件 type.h 第 598 行定义.

status

```
PreviewPngMapStatus agvc_interface::PreviewPngMapWorkingStatus::status
```

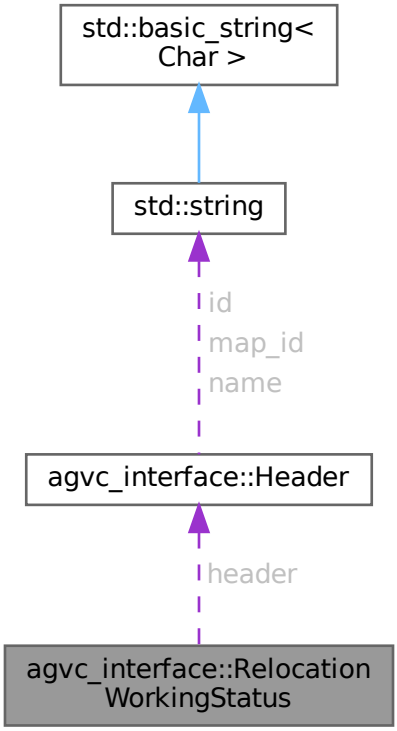
接口 previewPngMapFromAgv 的运行状态 NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 599 行定义.

该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.29 agvc_interface::RelocationWorkingStatus 结构体参考

接口Relocation 的工作状态
`#include <type.h>`
agvc_interface::RelocationWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的*Relocation* 指令 *id*
- RelocationStatus status

  接口*Relocation* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.29.1 详细描述

接口Relocation 的工作状态
在文件 type.h 第 516 行定义.

12.29.2 类成员变量说明

header

```
Header agvc_interface::RelocationWorkingStatus::header
```

header.id 代表下发的Relocation 指令 id
在文件 type.h 第 518 行定义.

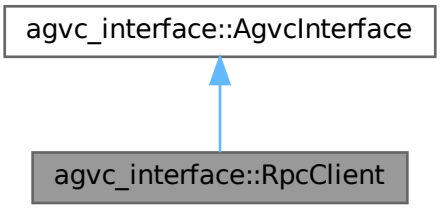
status

RelocationStatus agvc_interface::RelocationWorkingStatus::status
接口Relocation 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 519 行定义.
该结构体的文档由以下文件生成:

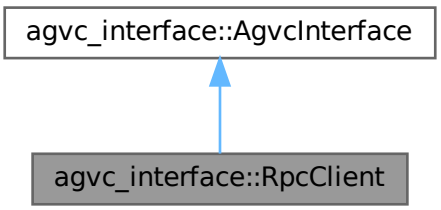
- include/agvc_interface/type.h

12.30 agvc_interface::RpcClient 类参考

RPC 客户端
`#include <rpc.h>`
类 agvc_interface::RpcClient 继承关系图:



agvc_interface::RpcClient 的协作图:



Public 类型

- enum Event { Connected , Disconnected }

Public 成员函数

- RpcClient (int mode=0)
  RpcClient
- ~RpcClient ()

- void setLogHandler (std::function< void(int, const char *, int, const std::string &)> handler)
  设置日志处理器
- int connect (const std::string &ip="", int port=0)
  连接到*RPC* 服务
- int disconnect ()
  断开*RPC* 连接
- bool hasConnected () const
  判断是否连接*RPC*
- int login (const std::string &usrname, const std::string &passwd)
  登录
- int logout ()
  登出
- bool hasLogined ()
  判断是否登录
- int setRequestTimeout (int timeout=10)
  设置*RPC* 请求超时时间
- int setEventHandler (std::function< void(int)> cb)
  设置事件处理
- int setExceptionFree (bool enable)
  是否关闭异常抛出
- int errorCode () const
  返回错误代码

Public 成员函数继承自 agvc_interface::AgvcInterface

- AgvcInterface ()
- virtual ~AgvcInterface ()
- int setSystemClock (const std::string &stamp)
  设置*AGV* 系统时间。时间戳，格式为*YYYY-MM-DDTHH:mm:ss.ssZ*，例如“*2017-04-15T11:40:03.12Z*”。
- AgvDetails getAgvDetails ()
- RunningInfo getRunningInfo ()
- Pose2d getAgvCurrentPose ()
- std::vector< Point2d > getLaserPointCloud ()
- AsyncInterfaceResultStatus getAsyncInterfaceResultStatus ()
  查询异步接口的运行情况，不代表异步接口本身的运行结果。异步接口：*saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv*。异步接口：*getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv*。异步接口：*getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv*。异步接口：*autoAlignRailway*。客户端与*AGV* 断联后，可调用该接口获取异步接口运行情况。返回值的每个字段中均有 *header*，其中 *header.id* 表示下发异步任务时的 *id* 号。*NONE*：异步接口未运行；*RUNNING*：异步接口运行中；*FINISH*：异步接口运行结束。
- std::vector< StationMark > getAllStations ()
  查询当前地图的所有站点信息。当前地图的所有站点信息。
- std::vector< StationMark > getAllStationsOfTargetMap (const Header &map_header)
  查询指定地图的所有站点信息。指定地图 *header(*命令 *id*，名称，时间，地图 *id)*。指定地图的所有站点信息。
- int addStation (const StationMark &station)
  添加或修改一个站点。待添加或修改的站点信息。*10100000*：添加或修改站点成功；*else*：添加或修改站点失败。
- int addStations (const std::vector< StationMark > &stations)
  添加或修改多个站点。待添加或修改的站点信息。*10100000*：添加或修改站点成功；*else*：添加或修改站点失败。
- int addCPStationUseChargingBoard (const Header &station_header, const double &dis_station_board=1.5)

通过自动识别充电桩位姿来添加充电站点。待添加或修改的充电站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。充电站点与充电桩之间的距离 *(*充电站点位于*AGV* 中心*)*, 单位: *m*, 范围*[1.0∼2.0]*。*10100000*: 添加充电站点成功; *else*: 添加充电站点失败。

- int deleteStation (const Header &station_header)

  删除指定站点, 会删除与该站点相关的路径。待删除站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除站点成功; *else*: 删除站点失败。

- int deleteStations (const std::vector< Header > &stations_header)

  删除多个站点, 会删除与站点相关的路径。待删除站点信息 *(*站点 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除站点成功; *else*: 删除站点失败。

- std::vector< PathStation > getAllPaths ()

  查询当前地图的所有路径信息。当前地图的所有路径信息。

- std::vector< PathStation > getAllPathsOfTargetMap (const Header &map_header)

  查询指定地图的所有路径信息。指定地图 *header(*命令 *id*, 名称, 时间, 地图 *id)*。指定地图的所有路径信息。

- PathStation getCurrentPath ()

  查询*AGV* 当前正在跟踪的路径信息。当前正在跟踪的路径信息。

- int generatePath (const PathStation &path_station)

  生成或修改一条路径。待生成或修改的路径信息。*10100000*: 修改或生成路径成功; *else*: 修改或生成路径失败。

- int generatePaths (const std::vector< PathStation > &paths_station)

  生成或修改多条路径。待生成或修改的路径信息。*10100000*: 修改或生成路径成功; *else*: 修改或生成路径失败。

- int deletePath (const Header &path_header)

  删除一条路径。待删除路径信息 *(*路径 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除路径成功; *else*: 删除路径失败。

- int deletePaths (const std::vector< Header > &paths_header)

  删除多条路径。待删除路径信息 *(*路径 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除路径成功; *else*: 删除路径失败。

- std::vector< Header > getMapList ()

  获取*AGV* 所有地图的信息头。所有地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。

- Header getCurrentMapHeader ()

  查询当前*AGV* 地图的信息头。当前地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。

- OccupancyGridMap getGridMapFromAgv (const Header &map_header)

  获取指定栅格地图信息。指定地图的信息头 *(*本次异步命令 *id*, 名称, 时间, 地图 *id)*。*map_id* 为*"current_map"* 时, 可特指为当前地图。指定栅格地图信息。

- Base64PngMap getBase64PngMapFromAgv (const Header &map_header)
- Base64PngMap previewPngMapFromAgv (const Header &map_header, const int &image_width_px=0, const int &image_height_px=0)
- int sendGridMapToAgv (const OccupancyGridMap &map)
- int sendBase64PngMapToAgv (const Base64PngMap &map)
- int saveMap (const Header &map_header)
- int switchMap (const Header &map_header)
- int deleteMap (const Header &map_header)

  删除一张指定地图。指定地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除地图成功; *else*: 删除地图失败。

- int deleteMaps (const std::vector< Header > &map_headers)

  删除多张指定地图。多个指定地图的信息头 *(*命令 *id*, 名称, 时间, 地图 *id)*。*10100000*: 删除地图成功; *else*: 删除地图失败。

- std::vector< MapVirtualArea > getAllMapVirtualArea ()

  查询当前地图的虚拟区域。当前地图的虚拟区域信息。

- std::vector< MapVirtualArea > getAllMapVirtualAreaOfTargetMap (const Header &map_header)

  查询目标地图的虚拟区域。指定地图 *header(*命令 *id*, 名称, 时间, 地图 *id)*。指定地图的所有虚拟区域。

- int addMapVirtualArea (const MapVirtualArea &map_virtual_area)

添加或修改一个虚拟区域。待添加或修改的虚拟区域信息。*10100000*: 添加或修改虚拟区域成功; *else*: 添加或修改虚拟区域失败。

- int addMapVirtualAreas (const std::vector< MapVirtualArea > &map_virtual_areas)

  添加或修改多个虚拟区域。待添加或修改的虚拟区域信息。*10100000*: 添加或修改虚拟区域成功; *else*: 添加或修改虚拟区域失败。
- int deleteMapVirtualArea (const Header &virtual_area_header)

  删除指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*, 虚拟区域名称, 时间, 所在地图 *id)*。*10100000*: 删除虚拟区域成功; *else*: 删除虚拟区域失败。
- int deleteMapVirtualAreas (const std::vector< Header > &virtual_areas_header)

  删除多个指定虚拟区域。指定虚拟区域信息头 *(*命令 *id*, 虚拟区域名称, 时间, 所在地图 *id)*。*10100000*: 删除虚拟区域成功; *else*: 删除虚拟区域失败。
- MapAllInfo getGridMapAllInfo (const Header &map_header)
- MapAllInfo getPngMapAllInfo (const Header &map_header)
- int setGridMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int setPngMapAllInfo (const MapAllInfo &map_all_info, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int deleteMapAllInfo (const Header &map_header)

  删除指定地图的全部信息, 包括地图数据、站点、路径、虚拟区信息。指定地图信息头 *(*命令 *id*, 地图名称, 时间, 地图 *id)*。*10100000*: 删除成功; *else*: 删除失败。
- std::vector< std::string > getWifiList ()
- int connectWifi (const std::string &ssid, const std::string &password)

  连接到指定 *WiFi*, 自动切换到 *WiFi* 模式。必须在有线连接时调用, 且执行时间较长。*WiFi* 名称。*WiFi* 密码。*10100000*: 连接 *WiFi* 成功; *else*: 连接 *WiFi* 失败。
- int enableHotspot (const std::string &password)

  开启热点, 自动切换到热点模式。必须在有线连接时调用, 且执行时间较长。热点密码; 如果为空使用 *”administrator”* 为默认密码, *SSID* 默认为 *SN* 码。*10100000*: 启用热点成功; *else*: 启用热点失败。
- std::vector< IpAddressInfo > getIpAddressList ()

  获取本机所有 *IP* 地址。本机所有 *IP* 地址。
- int changeRunningMode (const RunningMode &running_mode, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- int setControlSpeed (const Speed &speed)
- int setNavGoal (const NavGoalType &target)
- NavInfo getNavInfo ()
- int pauseAgvSpeed ()
- int resumeAgvSpeed ()
- int cancelNavigation ()
- int relocation (const Pose2d &init_pose, const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })
- std::string getAgvControllerParametersFile ()

  获取 *AGV* 控制器的参数文件。当前 *AGV* 控制器中的参数信息。
- int setAgvControllerParametersFile (const std::string &agv_parameters)
- int refreshAgvControllerParametersFile ()
- int resetAgvControllerParametersFile ()
- int checkPowerAndAutoCharge (const Header &check_power_header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &nav_board_charge_station_id=”99999”)
- int setLeaveBoardTargetStation (const Header &leave_board_target_station_header)

  设置自动下桩时的目标站点。自动下桩的站点 *header* 信息。*10100000*: 设置自动下桩站点成功; *else*: 设置自动下桩站点失败。
- Header getLeaveBoardTargetStation ()

  获取自动下桩时的目标站点。自动下桩时的目标站点 *Header* 信息。
- int forcedAgvToChargingBoard (const Header &forced_charge_header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &nav_board_charge_station_id=”99999”)
- int forcedAgvLeaveChargingBoard (const Header &forced_leave_header={ ”99999”, ”99999”, 99999, ”99999” }, const std::string &leave_board_target_station_id=”99999”)

- int sendAutoChargingCommand (const AutoChargingCommand &auto_charging_command)
- std::string getAgvLogMessage ()

  获取*AGV* 运行中的日志信息。*AGV* 运行中的日志信息。
- int updateSoftware (const std::string &upgrade_pack_path)

  AGV 软件升级。升级包路径，例如*"/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz"*。*10100000*: 目标版本升级成功；*10100201*: 正在升级中；*else*: 升级失败。
- std::vector< std::string > getSoftwareVersionList ()

  获取*AGV* 软件版本列表。软件版本列表集合 *{0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}*。
- int switchSoftwareVersion (const std::string &software_version)

  切换*AGV* 软件版本。软件版本，例如*"0.3.2+3b807d1-Linux_x86_64"*。*10100000*: 软件版本切换成功；*else*: 软件版本切换失败。
- int uninstallSoftware (const std::string &software_version)

  卸载*AGV* 软件版本。软件版本，例如*"0.3.2+3b807d1-Linux_x86_64"*。*10100000*: 软件版本卸载成功；*else*: 软件版本卸载失败。
- int updateFirmware (const FirmwareUpdateParam &update_firmware)

  AGV 固件更新。配置要更新固件的模块、固件存储路径。*10100000*: 下发固件更新信息成功；*else*: 固件正在更新中。
- FirmwareUpdateProcessInfo getUpdateFirmwareProcess ()

  获取固件更新过程信息。固件更新步骤信息 *(*步骤信息为 *failed* 代表更新失败*)*，更新进度 *(*进度信息为 *1.0* 代表更新成功*)*。
- int startCollectCalibrationData (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  开始采集数据。*LASER_ODOM* 标定采集数据量要求大于 *1000* 组，具体数据量可以在配置文件中修改。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*: 接口调用成功；*10120202*: 无控制权；*else*: 接口调用失败。
- int cancelCollectCalibrationData (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  取消采集数据。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*: 接口调用成功；*10120202*: 无控制权；*else*: 接口调用失败。
- int startCalibration (const CalibrationType &type, const Header &command_header={ "99999", "99999", 1, "99999" })

  开始标定。标定类型。*id* 字段代表下发本次命令的 *id*；其他字段无特殊含义。*10100000*: 接口调用成功；*10120202*: 无控制权；*else*: 接口调用失败。
- CalibrationProcessInfo getCalibrationProcessInfo (const CalibrationType &type)

  获取标定信息。标定类型。标定信息。
- int restartAgv ()

  重启*AGV*。*10100000*: 重启指令执行成功；*else*: 重启指令执行失败。
- int setAgvName (const std::string &agv_name)

  设置*AGV* 的名称。设置的*AGV* 名称。*10100000*: 设置名称成功；*else*: 设置名称失败。
- int setPriority (const std::string &name, const std::string &ip="")

  设置控制权。*1.*
- int releasePriority ()

  释放控制权。*10100000*: 释放控制权成功；*else*: 释放控制权失败。
- int scriptPaused ()

  暂停脚本。*10100000*: 脚本暂停成功；*else*: 脚本暂停失败。
- int scriptResume ()

  恢复脚本。*10100000*: 脚本恢复成功；*else*: 脚本恢复失败。
- int scriptStop ()

  停止脚本。*10100000*: 脚本停止成功；*else*: 脚本停止失败。
- ScriptRuntimeState getScriptStatus ()

  获取脚本运行状态。脚本运行状态。
- int setScriptStatus (ScriptRuntimeState status)

设置脚本运行状态。*10100000*：设置脚本运行状态成功；*else*：设置脚本运行状态失败。

- std::string errorCodeDecoder (const int &error_code)

  错误码查询。错误码。错误码含义；为空表示错误码未知。

- std::vector< Header > getCurrentErrorCodes ()

  获取当前的错误码。错误码集合。

- int soundLightPrompt ()

  寻找*AGV*，自动播放语音并特殊灯光闪烁。*10100000*：寻找指令下发成功；*else*：下发指令失败。

- bool isObstacleAhead (const double &detect_distance=1.0)

  AGV 行进方向是否有障碍物。检测距离，默认为 *1.0m*。*true*：行进方向存在障碍物；*false*：行进方向无障碍物。

- StationMark getNearestStation (const double &detect_distance=1.0)

  获取指定检测距离范围内的最近站点信息。检测距离，默认为 *1.0m*。最近站点信息，*id* 为*NONE* 代表检测距离内无站点。

- int autoAlignRailway (const Header &command_header={ ”99999”, ”99999”, 1, ”99999” })

12.30.1 详细描述

RPC 客户端
在文件 rpc.h 第 17 行定义.

12.30.2 成员枚举类型说明

Event

```
enum agvc_interface::RpcClient::Event
```

枚举值

Connected	
Disconnected	

在文件 rpc.h 第 20 行定义.

12.30.3 构造及析构函数说明

RpcClient()

```
agvc_interface::RpcClient::RpcClient (
          int mode = 0)
```

RpcClient

参数

mode	0-TCP 1-UDS

~RpcClient()

```
agvc_interface::RpcClient::~RpcClient ()
```

12.30.4 成员函数说明

connect()

```
int agvc_interface::RpcClient::connect (
          const std::string & ip = "",
          int port = 0)
```

连接到RPC 服务

参数

ip	IP 地址
port	端口号，RPC 的端口号是 30104
ip 和 *port* 为空时，采用 *unix*	domain sockets 通讯方式

返回值

0	RPC 连接成功
-8	RPC 连接失败，RPC 连接被拒绝
-15	RPC 连接失败，SDK 版本与Server 版本不兼容

disconnect()

```
int agvc_interface::RpcClient::disconnect ()
```

断开RPC 连接

返回值

0	成功
-1	失败

errorCode()

```
int agvc_interface::RpcClient::errorCode () const
```

返回错误代码

返回

hasConnected()

```
bool agvc_interface::RpcClient::hasConnected () const
```

判断是否连接RPC

返回值

true	已连接RPC
false	未连接RPC

hasLogined()

```
bool agvc_interface::RpcClient::hasLogined ()
```

判断是否登录

返回值

true	已登录
false	未登录

login()

```
int agvc_interface::RpcClient::login (
            const std::string & usrname,
            const std::string & passwd)
```

登录

参数

usrname	用户名
passwd	密码

返回

0

logout()

```
int agvc_interface::RpcClient::logout ()
```

登出

返回

0

setEventHandler()

```
int agvc_interface::RpcClient::setEventHandler (
            std::function< void(int)> cb)
```

设置事件处理

参数

cb	

返回

setExceptionFree()

```
int agvc_interface::RpcClient::setExceptionFree (
            bool enable)
```

是否关闭异常抛出

参数

enable	

返回

setLogHandler()

```
void agvc_interface::RpcClient::setLogHandler (
        std::function< void(int, const char *, int, const std::string &)> handler)
```

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。

Agvc SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Agvc SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

setRequestTimeout()

```
int agvc_interface::RpcClient::setRequestTimeout (
        int timeout = 10)
```

设置RPC 请求超时时间

参数

timeout	请求超时时间，单位 ms

返回

0

该类的文档由以下文件生成:

- include/agvc_interface/rpc.h

12.31 agvc_interface::RtdeClient 类参考

RTDE 客户端

`#include <rtde.h>`

Public 类型

- enum Event { Connected , Disconnected }

Public 成员函数

- RtdeClient (int mode=0)
  RTDE 客户端初始化
- ~RtdeClient ()
- void setLogHandler (std::function< void(int, const char *, int, const std::string &)> handler)
  设置日志处理器
- int connect (const std::string &ip="", int port=0)
  连接到服务器
- bool hasConnected () const
  socket 是否已连接
- bool hasConnected1 (std::function< void(bool)> callback)
  socket 是否已连接
- int login (const std::string &usrname, const std::string &passwd)
  登录
- bool hasLogined ()
  是否已经登录
- int logout ()
  登出
- int disconnect ()
  断开连接
- int getProtocolVersion ()
  获取协议版本号
- std::map< std::string, int > getInputMaps ()
  获取输入列表
- std::map< std::string, int > getOutputMaps ()
  获取输出列表
- int setTopic (bool to_server, const std::vector< std::string > &names, double freq, int expected↩ _chanel)
  设置话题
- int removeTopic (bool to_server, int chanel)
  取消订阅
- std::unordered_map< int, agvc_interface::RtdeRecipe > getRegisteredInputRecipe ()
  获取已注册的输入菜单
- std::unordered_map< int, agvc_interface::RtdeRecipe > getRegisteredOutputRecipe ()
  获取已注册的输出菜单
- int subscribe (int chanel, std::function< void(InputParser &)> callback)

订阅 *subscribe from output*

- int publish (int chanel, std::function< void(OutputBuilder &)> callback)
  发布 *publish to input*
- int setEventHandler (std::function< void(int)> cb)
  设置事件处理

Private 属性

- Impl * impl

12.31.1 详细描述

RTDE 客户端
在文件 rtde.h 第 79 行定义.

12.31.2 成员枚举类型说明

Event

```
enum agvc_interface::RtdeClient::Event
```

枚举值

Connected	
Disconnected	

在文件 rtde.h 第 82 行定义.

12.31.3 构造及析构函数说明

RtdeClient()

```
agvc_interface::RtdeClient::RtdeClient (
          int mode = 0)
```

RTDE 客户端初始化

参数

mode	设置通讯方式，0 tcp 通讯 1 uds 通讯

~RtdeClient()

```
agvc_interface::RtdeClient::~RtdeClient ()
```

12.31.4 成员函数说明

connect()

```
int agvc_interface::RtdeClient::connect (
          const std::string & ip = "",
          int port = 0)
```

连接到服务器

参数

ip	IP 地址
port	端口号，RTDE 端口号为 30110

返回值

0	连接成功
1	在执行函数前，已连接
-1	连接失败

disconnect()

```
int agvc_interface::RtdeClient::disconnect ()
```

断开连接

返回值

0	成功
-1	失败

getInputMaps()

```
std::map< std::string, int > agvc_interface::RtdeClient::getInputMaps ()
```

获取输入列表

返回

输入列表

getOutputMaps()

```
std::map< std::string, int > agvc_interface::RtdeClient::getOutputMaps ()
```

获取输出列表

返回

输出列表

getProtocolVersion()

```
int agvc_interface::RtdeClient::getProtocolVersion ()
```

获取协议版本号

返回

协议版本号

getRegisteredInputRecipe()

```
std::unordered_map< int, agvc_interface::RtdeRecipe > agvc_interface::RtdeClient::getRegisteredInputRecipe
()
```

获取已注册的输入菜单

返回

已注册的输入菜单

getRegisteredOutputRecipe()

```
std::unordered_map< int, agvc_interface::RtdeRecipe > agvc_interface::RtdeClient::getRegisteredOutput↩
Recipe ()
```

获取已注册的输出菜单

返回

已注册的输出菜单

hasConnected()

```
bool agvc_interface::RtdeClient::hasConnected () const
```

socket 是否已连接

返回值

true	已连接 socket
false	未连接 socket

hasConnected1()

```
bool agvc_interface::RtdeClient::hasConnected1 (
            std::function< void(bool)> callback)
```

socket 是否已连接

参数

callback	

返回值

true	已连接 socket
false	未连接 socket

hasLogined()

```
bool agvc_interface::RtdeClient::hasLogined ()
```

是否已经登录

返回值

true	已登录
false	未登录

login()

```
int agvc_interface::RtdeClient::login (
            const std::string & usrname,
            const std::string & passwd)
```

登录

参数

usrname	用户名
passwd	密码

返回值

0	成功
-1	失败

logout()

```
int agvc_interface::RtdeClient::logout ()
```

登出

返回

0

publish()

```
int agvc_interface::RtdeClient::publish (
            int chanel,
            std::function< void(OutputBuilder &)> callback)
```

发布 publish to input

参数

chanel	通道
callback	回调函数，用于构建发布的输出信息。 回调函数的定义如下：void callback(OutputBuilder &builder)

返回值

0	成功
-1	失败

removeTopic()

```
int agvc_interface::RtdeClient::removeTopic (
          bool to_server,
          int chanel)
```

取消订阅

参数

to_server	数据流向 true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
chanel	通道

返回值

0	成功
1	失败

setEventHandler()

```
int agvc_interface::RtdeClient::setEventHandler (
          std::function< void(int)> cb)
```

设置事件处理

参数

cb	

返回

setLogHandler()

```
void agvc_interface::RtdeClient::setLogHandler (
          std::function< void(int, const char *, int, const std::string &)> handler)
```

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。

Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

setTopic()

```
int agvc_interface::RtdeClient::setTopic (
          bool to_server,
          const std::vector< std::string > & names,
          double freq,
          int expected_chanel)
```

设置话题

参数

to_server	数据流向。true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
names	服务器推送的信息列表
freq	服务器推送信息的频率
expected_chanel	通道。取值范围：0~99，发布不同的话题走不同的通道

返回值

expected_chanel 参数的值	成功
-1	失败

subscribe()

```
int agvc_interface::RtdeClient::subscribe (
          int chanel,
          std::function< void(InputParser &)> callback)
```

订阅 subscribe from output

参数

chanel	通道
callback	回调函数，用于处理订阅的输入信息。 回调函数的定义如下：void callback(InputParser &parser)

返回

0

12.31.5 类成员变量说明

impl

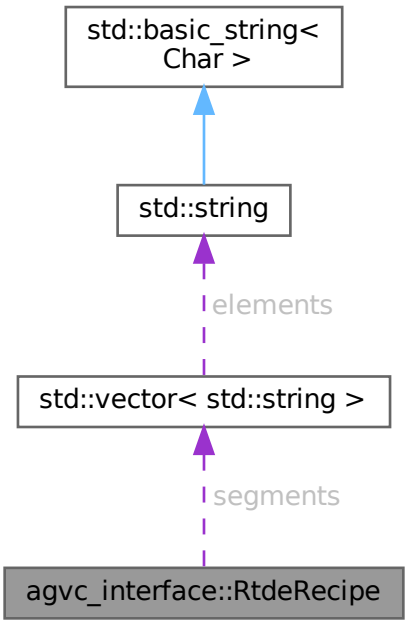
```
Impl* agvc_interface::RtdeClient::impl  [private]
```

在文件 rtde.h 第 282 行定义.
该类的文档由以下文件生成:

- include/agvc_interface/rtde.h

12.32 agvc_interface::RtdeRecipe 结构体参考

RTDE 菜单
`#include <type.h>`
agvc_interface::RtdeRecipe 的协作图:



Public 属性

- bool to_server

  输入/输出
- int chanel

  通道
- double frequency

  更新频率
- int trigger

  触发方式 *(*该功能暂未实现*): 0* - 周期*; 1* - 变化
- std::vector< std::string > segments

  字段列表

12.32.1 详细描述

RTDE 菜单
在文件 type.h 第 841 行定义.

12.32.2 类成员变量说明

chanel

```
int agvc_interface::RtdeRecipe::chanel
```

通道
在文件 type.h 第 844 行定义.

frequency

```
double agvc_interface::RtdeRecipe::frequency
```

更新频率
在文件 type.h 第 845 行定义.

segments

```
std::vector<std::string> agvc_interface::RtdeRecipe::segments
```

字段列表
在文件 type.h 第 847 行定义.

to_server

```
bool agvc_interface::RtdeRecipe::to_server
```

输入/输出
在文件 type.h 第 843 行定义.

trigger

```
int agvc_interface::RtdeRecipe::trigger
```

触发方式 (该功能暂未实现): 0 - 周期; 1 - 变化
在文件 type.h 第 846 行定义.
该结构体的文档由以下文件生成:

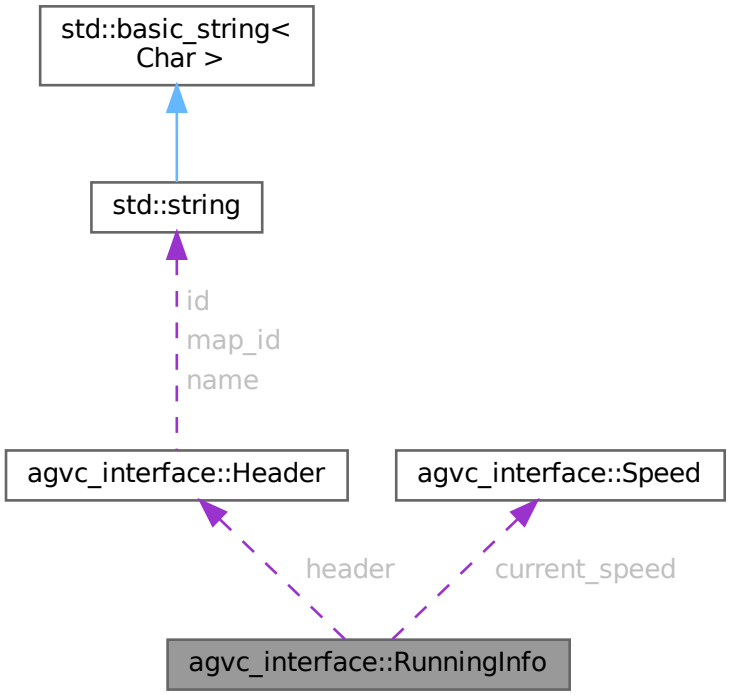
- include/agvc_interface/type.h

12.33 agvc_interface::RunningInfo 结构体参考

agv 运行信息

`#include <type.h>`

agvc_interface::RunningInfo 的协作图:



Public 属性

- Header header
  agv 的*SN* 码，*agv* 的名称，当前时间，地图的 *id*
- RunningMode running_mode
  agv 当前运行模式
- Speed current_speed
  当前行驶速度 *{v w}*，单位 *{m/s rad/s}*
- int8_t location_score
  当前定位评分，*[0-100]*
- double total_dist
  累计行驶里程，单位 *m*
- double current_dist
  本次累计行驶里程，单位 *m*
- double total_running_time
  累计运行时间，单位 *s*
- double current_running_time
  本次运行时间，单位 *s*
- double battery_voltage
  电池电压，单位 *V*

- double battery_current
  电池电流，单位 *A*
- int8_t remaining_voltage
  剩余电量，百分比% *[0-100]*
- int8_t battery_status
  电池状态，*0*-无,*1*-放电，*2*-充电，*3*-充满，*4*-急需充电
- int8_t safety_edge_status
  安全触边状态，*0*-无,*1*-左前侧碰撞，*2*-右前碰撞，*3*-左后碰撞，*4*-右后碰撞
- bool low_battery
  是否电量低
- bool pause
  是否处于暂停状态
- bool emergency_stop
  是否处于急停状态
- bool blocked
  是否被障碍物遮挡
- int8_t obstacle_level
  停障等级，*0*-未停障，*1*-近距离停障，*2*-中等距离停障，*3*-远距离停障
- bool localization_loss
  是否定位丢失
- bool fault_alarm
  故障报警（电机故障、传感器故障、电池故障），具体故障信息可在日志中查看
- bool break_release_flag
  是否释放刹车，*true* 代表释放，*false* 代表未释放

12.33.1 详细描述

agv 运行信息
在文件 type.h 第 460 行定义.

12.33.2 类成员变量说明

battery_current

```
double agvc_interface::RunningInfo::battery_current
```

电池电流，单位 A
在文件 type.h 第 471 行定义.

battery_status

```
int8_t agvc_interface::RunningInfo::battery_status
```

电池状态，0-无,1-放电，2-充电，3-充满，4-急需充电
在文件 type.h 第 473 行定义.

battery_voltage

```
double agvc_interface::RunningInfo::battery_voltage
```

电池电压，单位 V
在文件 type.h 第 470 行定义.

blocked

```
bool agvc_interface::RunningInfo::blocked
```

是否被障碍物遮挡
在文件 type.h 第 478 行定义.

break_release_flag

bool agvc_interface::RunningInfo::break_release_flag
是否释放刹车，true 代表释放，false 代表未释放
在文件 type.h 第 482 行定义.

current_dist

double agvc_interface::RunningInfo::current_dist
本次累计行驶里程，单位 m
在文件 type.h 第 467 行定义.

current_running_time

double agvc_interface::RunningInfo::current_running_time
本次运行时间，单位 s
在文件 type.h 第 469 行定义.

current_speed

Speed agvc_interface::RunningInfo::current_speed
当前行驶速度 {v w}，单位 {m/s rad/s}
在文件 type.h 第 464 行定义.

emergency_stop

bool agvc_interface::RunningInfo::emergency_stop
是否处于急停状态
在文件 type.h 第 477 行定义.

fault_alarm

bool agvc_interface::RunningInfo::fault_alarm
故障报警（电机故障、传感器故障、电池故障）, 具体故障信息可在日志中查看
在文件 type.h 第 481 行定义.

header

Header agvc_interface::RunningInfo::header
agv 的SN 码，agv 的名称，当前时间，地图的 id
在文件 type.h 第 462 行定义.

localization_loss

bool agvc_interface::RunningInfo::localization_loss
是否定位丢失
在文件 type.h 第 480 行定义.

location_score

int8_t agvc_interface::RunningInfo::location_score
当前定位评分，[0-100]
在文件 type.h 第 465 行定义.

low_battery

bool agvc_interface::RunningInfo::low_battery
是否电量低
在文件 type.h 第 475 行定义.

obstacle_level

```
int8_t agvc_interface::RunningInfo::obstacle_level
```

停障等级，0-未停障，1-近距离停障，2-中等距离停障，3-远距离停障
在文件 type.h 第 479 行定义.

pause

```
bool agvc_interface::RunningInfo::pause
```

是否处于暂停状态
在文件 type.h 第 476 行定义.

remaining_voltage

```
int8_t agvc_interface::RunningInfo::remaining_voltage
```

剩余电量，百分比% [0-100]
在文件 type.h 第 472 行定义.

running_mode

```
RunningMode agvc_interface::RunningInfo::running_mode
```

agv 当前运行模式
在文件 type.h 第 463 行定义.

safety_edge_status

```
int8_t agvc_interface::RunningInfo::safety_edge_status
```

安全触边状态，0-无,1-左前侧碰撞，2-右前碰撞，3-左后碰撞，4-右后碰撞
在文件 type.h 第 474 行定义.

total_dist

```
double agvc_interface::RunningInfo::total_dist
```

累计行驶里程，单位 m
在文件 type.h 第 466 行定义.

total_running_time

```
double agvc_interface::RunningInfo::total_running_time
```

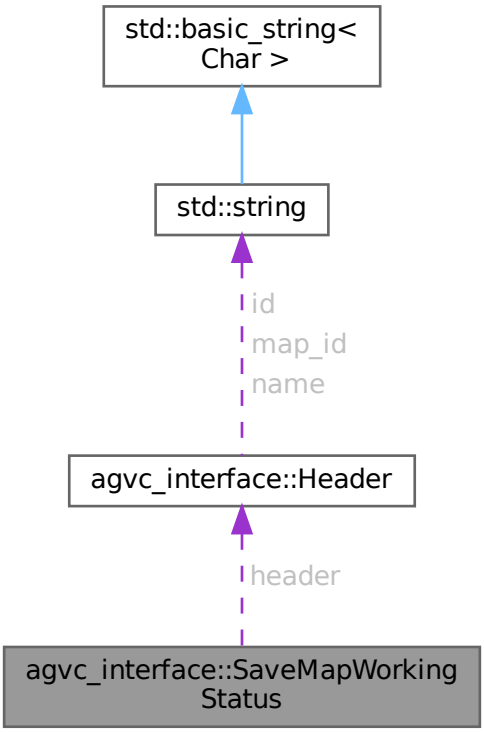
累计运行时间，单位 s
在文件 type.h 第 468 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.34 agvc_interface::SaveMapWorkingStatus 结构体参考

接口 saveMap 的工作状态
`#include <type.h>`

agvc_interface::SaveMapWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *saveMap* 指令 *id*
- SaveMapStatus status

  接口*SaveMap* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.34.1 详细描述

接口 saveMap 的工作状态
在文件 type.h 第 488 行定义.

12.34.2 类成员变量说明

header

```
Header agvc_interface::SaveMapWorkingStatus::header
```

header.id 代表下发的 saveMap 指令 id
在文件 type.h 第 490 行定义.

status

```
SaveMapStatus agvc_interface::SaveMapWorkingStatus::status
```

接口SaveMap 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 491 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.35 agvc_interface::ScriptClient 类参考

SCRIPT 客户端
`#include <script.h>`

Public 类型

- enum Event { Connected , Disconnected }

Public 成员函数

- ScriptClient (int mode=0)
- ~ScriptClient ()
- void setLogHandler (std::function< void(int, const char *, int, const std::string &)> handler)
  设置日志处理器
- int connect (const std::string &ip="", int port=0)
  连接到服务器
- bool hasConnected () const
  是否处于连接状态
- int login (const std::string &usrname, const std::string &passwd)
  登录
- bool hasLogined ()
  返回客户端是否登录
- int logout ()
  登出
- int disconnect ()
  断开连接
- int sendFile (const std::string &path)
  发送脚本文件
- int sendString (const std::string &script)
  发送脚本内容
- int send (const std::string &chunck_name, std::function< int(ScriptWriter &)> cb)
  使用*ScriptWriter* 构建服务器脚本
- int subscribeVariableUpdate (std::function< void(const std::string &, const std::string &)> cb)
  设置服务器脚本的全局变量
- int subscribeScriptError (std::function< void(const std::string &)> cb)
  设置服务器脚本的错误码
- int subscribeScriptError2 (std::function< void(const std::string &, const std::string &)> cb)
- int setEventHandler (std::function< void(int)> cb)
  设置事件处理

Private 属性

- Impl * impl

12.35.1 详细描述

SCRIPT 客户端
在文件 script.h 第 31 行定义.

12.35.2 成员枚举类型说明

Event

`enum agvc_interface::ScriptClient::Event`

枚举值

Connected	
Disconnected	

在文件 script.h 第 34 行定义.

12.35.3 构造及析构函数说明

ScriptClient()

```
agvc_interface::ScriptClient::ScriptClient (
          int mode = 0)
```

~ScriptClient()

```
agvc_interface::ScriptClient::~ScriptClient ()
```

12.35.4 成员函数说明

connect()

```
int agvc_interface::ScriptClient::connect (
          const std::string & ip = "",
          int port = 0)
```

连接到服务器

参数

ip	IP 地址
port	端口号。SCRIPT 端口号为 30004

返回值

0	连接成功
1	在执行函数前，已连接
-1	连接失败

disconnect()

```
int agvc_interface::ScriptClient::disconnect ()
```

断开连接

返回值

0	成功
-1	失败

hasConnected()

```
bool agvc_interface::ScriptClient::hasConnected () const
```

是否处于连接状态

返回值

true	已连接
false	未连接

hasLogined()

```
bool agvc_interface::ScriptClient::hasLogined ()
```

返回客户端是否登录

返回值

true	已登录
false	未登录

login()

```
int agvc_interface::ScriptClient::login (
          const std::string & usrname,
          const std::string & passwd)
```

登录

参数

usrname	用户名
passwd	密码

返回值

0	成功
-1	失败

logout()

```
int agvc_interface::ScriptClient::logout ()
```

登出

返回

0

send()

```
int agvc_interface::ScriptClient::send (
            const std::string & chunck_name,
            std::function< int(ScriptWriter &)> cb)
```

使用ScriptWriter 构建服务器脚本

参数

chunck_name	
cb	

返回值

0	成功
-1	失败

sendFile()

```
int agvc_interface::ScriptClient::sendFile (
            const std::string & path)
```

发送脚本文件
远程调用机器人的脚本

参数

path	文件在机器人端的路径

返回值

0	成功
-1	失败

sendString()

```
int agvc_interface::ScriptClient::sendString (
            const std::string & script)
```

发送脚本内容
调用本地的脚本

参数

script	脚本内容

返回值

0	成功
-1	失败

setEventHandler()

```
int agvc_interface::ScriptClient::setEventHandler (
          std::function< void(int)> cb)
```

设置事件处理

参数

cb	

返回

setLogHandler()

```
void agvc_interface::ScriptClient::setLogHandler (
          std::function< void(int, const char *, int, const std::string &)> handler)
```

设置日志处理器

此函数可设置自定义的日志处理函数来处理日志消息。
Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。

注解

setLogHandler 函数要放在即将触发的日志之前，否则会按照默认的形式输出日志。

参数

handler	日志处理函数 此日志处理函数的下定义如下: void handler(int level, const char* filename, int line, const std::string& message) level 表示日志等级 0: LOGLEVEL_FATAL 严重的错误 1: LOGLEVEL_ERROR 错误 2: LOGLEVEL_WARNING 警告 3: LOGLEVEL_INFO 通知 4: LOGLEVEL_DEBUG 调试 5: LOGLEVEL_BACKTRACE 跟踪 filename 表示文件名 line 表示代码行号 message 表示日志信息

返回

无

subscribeScriptError()

```
int agvc_interface::ScriptClient::subscribeScriptError (
          std::function< void(const std::string &)> cb)
```

设置服务器脚本的错误码

参数

cb	脚本的错误码

返回

0

subscribeScriptError2()

```
int agvc_interface::ScriptClient::subscribeScriptError2 (
          std::function< void(const std::string &, const std::string &)> cb)
```

subscribeVariableUpdate()

```
int agvc_interface::ScriptClient::subscribeVariableUpdate (
          std::function< void(const std::string &, const std::string &)> cb)
```

设置服务器脚本的全局变量

参数

cb	脚本的全局变量

返回

0

12.35.5 类成员变量说明

impl

```
Impl* agvc_interface::ScriptClient::impl  [private]
```

在文件 script.h 第 186 行定义.
该类的文档由以下文件生成:

- include/agvc_interface/script.h

12.36 agvc_interface::ScriptWriter 类参考

```
#include <script.h>
```

Public 成员函数

- virtual ~ScriptWriter ()=default
- virtual ScriptWriter & append (const std::string &line)=0
- virtual ScriptWriter & append (const char *buf, size_t len)=0
- virtual ScriptWriter & moveJoint ()=0
- virtual ScriptWriter & moveLine ()=0
- virtual ScriptWriter & ifCondition ()=0
- virtual ScriptWriter & elseCondition ()=0
- virtual ScriptWriter & elseIfCondition ()=0
- virtual ScriptWriter & whileCondition ()=0
- virtual ScriptWriter & end ()=0

12.36.1 详细描述

在文件 script.h 第 14 行定义.

12.36.2 构造及析构函数说明

~ScriptWriter()

```
virtual agvc_interface::ScriptWriter::~ScriptWriter ()  [virtual], [default]
```

12.36.3 成员函数说明

append() [1/2]

```
virtual ScriptWriter & agvc_interface::ScriptWriter::append (
            const char * buf,
            size_t len)  [pure virtual]
```

append() [2/2]

```
virtual ScriptWriter & agvc_interface::ScriptWriter::append (
            const std::string & line)  [pure virtual]
```

elseCondition()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::elseCondition ()  [pure virtual]
```

elseIfCondition()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::elseIfCondition ()  [pure virtual]
```

end()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::end ()  [pure virtual]
```

ifCondition()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::ifCondition ()  [pure virtual]
```

moveJoint()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::moveJoint ()  [pure virtual]
```

moveLine()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::moveLine ()  [pure virtual]
```

whileCondition()

```
virtual ScriptWriter & agvc_interface::ScriptWriter::whileCondition ()  [pure virtual]
```

该类的文档由以下文件生成:

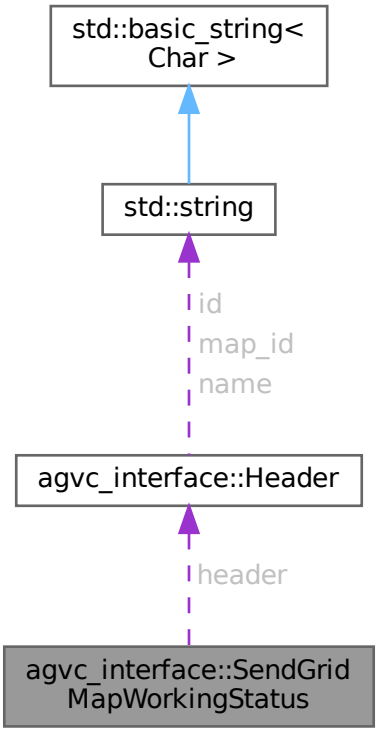
- include/agvc_interface/script.h

12.37 agvc_interface::SendGridMapWorkingStatus 结构体参考

接口 sendGridMapToAgv 的工作状态

`#include <type.h>`

agvc_interface::SendGridMapWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *sendGridMapToAgv* 指令 *id*

- SendGridMapStatus status

  接口 *sendGridMapToAgv* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.37.1 详细描述

接口 sendGridMapToAgv 的工作状态

在文件 type.h 第 552 行定义.

12.37.2 类成员变量说明

header

`Header agvc_interface::SendGridMapWorkingStatus::header`
header.id 代表下发的 sendGridMapToAgv 指令 id
在文件 type.h 第 554 行定义.

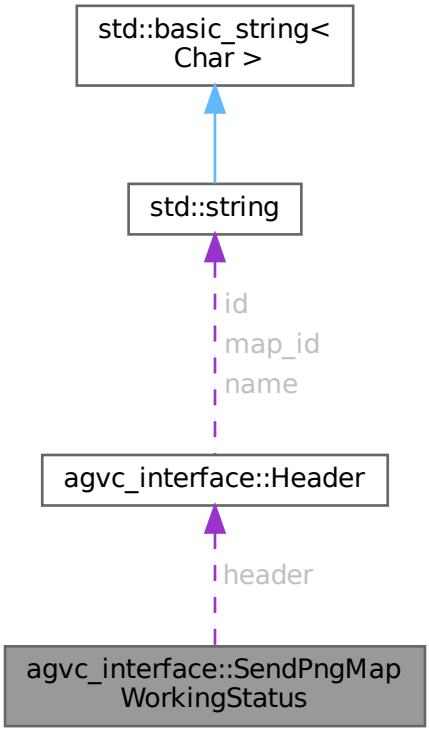
status

`SendGridMapStatus agvc_interface::SendGridMapWorkingStatus::status`
接口 sendGridMapToAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 555 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.38 agvc_interface::SendPngMapWorkingStatus 结构体参考

接口 sendBase64PngMapToAgv 的工作状态
`#include <type.h>`
agvc_interface::SendPngMapWorkingStatus 的协作图:



Public 属性

- Header header
  header.id 代表下发的 *sendBase64PngMapToAgv* 指令 *id*
- SendPngMapStatus status

接口 *sendBase64PngMapToAgv* 的运行状态，*NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.38.1 详细描述

接口 sendBase64PngMapToAgv 的工作状态

在文件 type.h 第 567 行定义.

12.38.2 类成员变量说明

header

```
Header agvc_interface::SendPngMapWorkingStatus::header
```

header.id 代表下发的 sendBase64PngMapToAgv 指令 id

在文件 type.h 第 569 行定义.

status

```
SendPngMapStatus agvc_interface::SendPngMapWorkingStatus::status
```

接口 sendBase64PngMapToAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束

在文件 type.h 第 570 行定义.

该结构体的文档由以下文件生成:

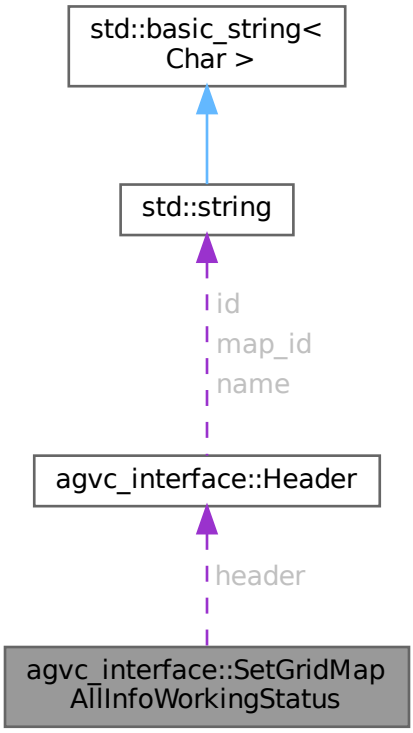
- include/agvc_interface/type.h

12.39 agvc_interface::SetGridMapAllInfoWorkingStatus 结构体参考

接口 setGridMapAllInfo 的工作状态

```
#include <type.h>
```

agvc_interface::SetGridMapAllInfoWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *setGridMapAllInfo* 指令 *id*
- SetGridMapAllInfoStatus status

  接口 *setGridMapAllInfo* 的运行状态 *NONE* -接口未执行 *RUNNING*-执行中，*FINISH*-执行结束

12.39.1 详细描述

接口 setGridMapAllInfo 的工作状态
在文件 type.h 第 576 行定义.

12.39.2 类成员变量说明

header

```
Header agvc_interface::SetGridMapAllInfoWorkingStatus::header
```

header.id 代表下发的 setGridMapAllInfo 指令 id
在文件 type.h 第 578 行定义.

status

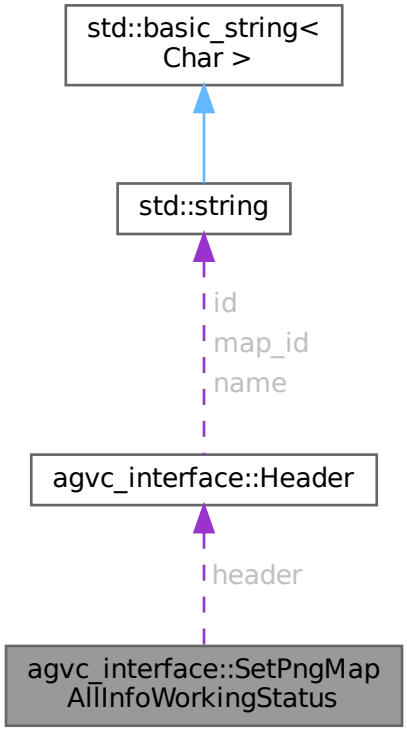
```
SetGridMapAllInfoStatus agvc_interface::SetGridMapAllInfoWorkingStatus::status
```

接口 setGridMapAllInfo 的运行状态 NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 579 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.40 agvc_interface::SetPngMapAllInfoWorkingStatus 结构体参考

接口 setPngMapAllInfo 的工作状态
`#include <type.h>`
agvc_interface::SetPngMapAllInfoWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *setPngMapAllInfo* 指令 *id*
- SetPngMapAllInfoStatus status

  接口 *setPngMapAllInfo* 的运行状态， *NONE* -接口未执行 *RUNNING*-执行中， *FINISH*-执行结束

12.40.1 详细描述

接口 setPngMapAllInfo 的工作状态
在文件 type.h 第 525 行定义.

12.40.2 类成员变量说明

header

```
Header agvc_interface::SetPngMapAllInfoWorkingStatus::header
```

header.id 代表下发的 setPngMapAllInfo 指令 id
在文件 type.h 第 527 行定义.

status

SetPngMapAllInfoStatus agvc_interface::SetPngMapAllInfoWorkingStatus::status
接口 setPngMapAllInfo 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 528 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.41 agvc_interface::Speed 结构体参考

速度
`#include <type.h>`

Public 属性

- double v

  单位 *m/s*
- double w

  单位 *rad/s*

12.41.1 详细描述

速度
在文件 type.h 第 416 行定义.

12.41.2 类成员变量说明

v

`double agvc_interface::Speed::v`
单位 m/s
在文件 type.h 第 418 行定义.

w

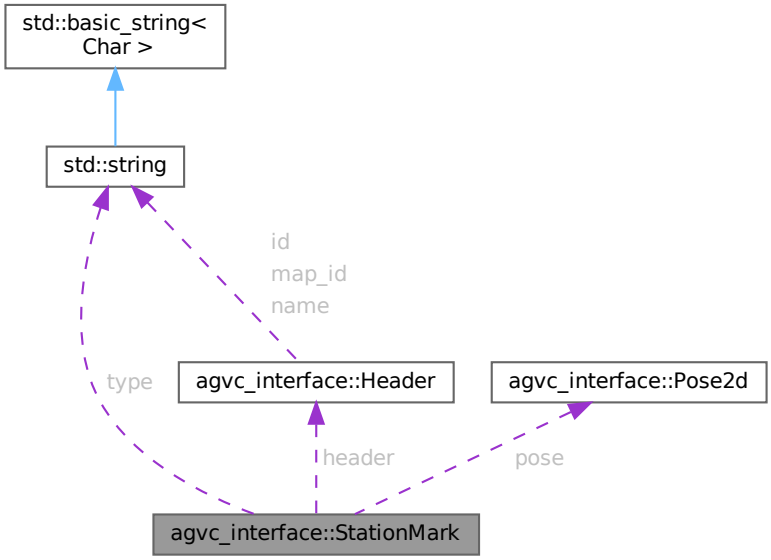
`double agvc_interface::Speed::w`
单位 rad/s
在文件 type.h 第 419 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.42 agvc_interface::StationMark 结构体参考

站点
`#include <type.h>`

agvc_interface::StationMark 的协作图:



Public 属性

- Header header

  站点 *id*，站点名称，当前时间，站点所在地图的 *id*
- std::string type

  普通站点是*"ST"*, 充电站点“*CP*”
- Pose2d pose

  站点位姿，*{x(m) y(m) theta(rad)}*

12.42.1 详细描述

站点
在文件 type.h 第 641 行定义.

12.42.2 类成员变量说明

header

```
Header agvc_interface::StationMark::header
```

站点 id，站点名称，当前时间，站点所在地图的 id
在文件 type.h 第 643 行定义.

pose

```
Pose2d agvc_interface::StationMark::pose
```

站点位姿，{x(m) y(m) theta(rad)}
在文件 type.h 第 645 行定义.

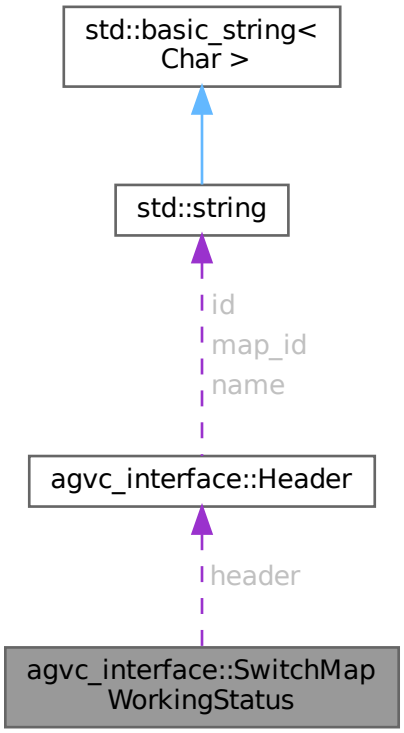
type

`std::string agvc_interface::StationMark::type`
普通站点是"ST", 充电站点 "CP"
在文件 type.h 第 644 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

12.43 agvc_interface::SwitchMapWorkingStatus 结构体参考

接口 switchMap 的工作状态
`#include <type.h>`
agvc_interface::SwitchMapWorkingStatus 的协作图:



Public 属性

- Header header

  header.id 代表下发的 *switchMap* 指令 *id*
- SwitchMapStatus status

  接口 *switchMap* 的运行状态, *NONE* -接口未执行 *RUNNING*-执行中, *FINISH*-执行结束

12.43.1 详细描述

接口 switchMap 的工作状态
在文件 type.h 第 497 行定义.

12.43.2 类成员变量说明

header

Header agvc_interface::SwitchMapWorkingStatus::header
header.id 代表下发的 switchMap 指令 id
在文件 type.h 第 499 行定义.

status

SwitchMapStatus agvc_interface::SwitchMapWorkingStatus::status
接口 switchMap 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
在文件 type.h 第 500 行定义.
该结构体的文档由以下文件生成:

- include/agvc_interface/type.h

Chapter 13

文件说明

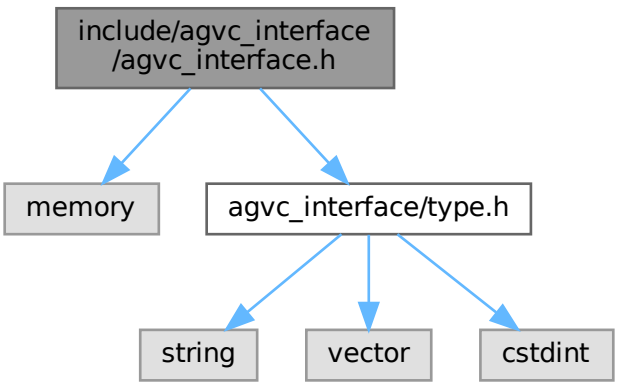
13.1 doc/deprecated_zh.md 文件参考

13.2 doc/SDK_overview.md 文件参考

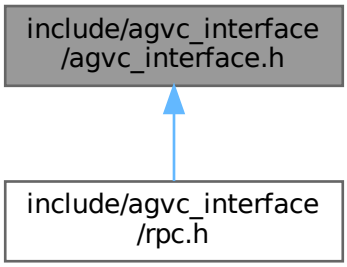
13.3 include/agvc_interface/agvc_interface.h 文件参考

```
#include <memory>
#include "agvc_interface/type.h"
```

agvc_interface.h 的引用 (Include) 关系图:



此图展示该文件被哪些文件直接或间接地引用了:



类

- class agvc_interface::AgvcInterface

命名空间

- namespace agvc_interface

宏定义

- #define INTERFACE_VERSION_MAJOR 0
- #define INTERFACE_VERSION_MINOR 4
- #define INTERFACE_VERSION_PATCH 0
- #define INTERFACE_VERSION (INTERFACE_VERSION_MAJOR * 1000000 + INTERFACE_VERSION_MI * 1000 + INTERFACE_VERSION_PATCH)
- #define AgvcInterface_DECLARES

类型定义

- using agvc_interface::AgvcInterfacePtr = std::shared_ptr<AgvcInterface>

13.3.1 宏定义说明

AgvcInterface_DECLARES

```
#define AgvcInterface_DECLARES
```

在文件 agvc_interface.h 第 1496 行定义.

INTERFACE_VERSION

```
#define INTERFACE_VERSION (INTERFACE_VERSION_MAJOR * 1000000 + INTERFACE_VERSION_MINOR * 1000 + INTERFACE_VERSION_PATCH)
```

在文件 agvc_interface.h 第 10 行定义.

INTERFACE_VERSION_MAJOR

```
#define INTERFACE_VERSION_MAJOR 0
```

在文件 agvc_interface.h 第 7 行定义.

INTERFACE_VERSION_MINOR

```
#define INTERFACE_VERSION_MINOR 4
```

在文件 agvc_interface.h 第 8 行定义.

INTERFACE_VERSION_PATCH

```
#define INTERFACE_VERSION_PATCH 0
```

在文件 agvc_interface.h 第 9 行定义.

13.4 agvc_interface.h

浏览该文件的文档.

```
00001 #ifndef AGVC_INTERFACE_H
00002 #define AGVC_INTERFACE_H
00003
00004 #include <memory>
00005 #include "agvc_interface/type.h"
00006
00007 #define INTERFACE_VERSION_MAJOR 0
00008 #define INTERFACE_VERSION_MINOR 4
00009 #define INTERFACE_VERSION_PATCH 0
00010 #define INTERFACE_VERSION       (INTERFACE_VERSION_MAJOR * 1000000 + INTERFACE_VERSION_MINOR * 1000 +
      INTERFACE_VERSION_PATCH)
00011
00012 /**
00013  * \chinese
00014  * @defgroup Map Map (地图模块)
00015  * \endchinese
00016  * \english
00017  * @defgroup Map Map
00018  * \endenglish
00019  * @brief
00020  * @zh AGVC 自定义地图的创建、切换、删除、数据读写等核心功能
00021  * @en Core functions for creating, switching, deleting, reading, and writing AGVC custom maps.
00022  */
00023
00024 /**
00025  * \chinese
00026  * @defgroup Station Station (站点模块)
00027  * \endchinese
00028  * \english
00029  * @defgroup Station Station
00030  * \endenglish
00031  * @brief
00032  * @zh AGVC 地图站点的增删改查、充电站点自动识别等功能
00033  * @en Functions for adding, deleting, modifying, querying AGVC map stations, and auto-detecting charging stations.
00034  */
00035
00036 /**
00037  * \chinese
00038  * @defgroup Path Path (路径模块)
00039  * \endchinese
00040  * \english
00041  * @defgroup Path Path
00042  * \endenglish
00043  * @brief
00044  * @zh AGVC 地图路径的生成、修改、删除、查询等功能
00045  * @en Functions for generating, modifying, deleting, and querying AGVC map paths.
00046  */
00047
00048 /**
00049  * \chinese
00050  * @defgroup Navigation Navigation (导航模块)
00051  * \endchinese
00052  * \english
00053  * @defgroup Navigation Navigation
00054  * \endenglish
00055  * @brief
00056  * @zh AGVC 运动控制、导航任务下发、重定位、模式切换等功能
00057  * @en Functions for AGVC motion control, navigation goals, relocation, and mode switching.
00058  */
00059
00060 /**
00061  * \chinese
00062  * @defgroup Charging Charging (充电模块)
00063  * \endchinese
00064  * \english
00065  * @defgroup Charging Charging
00066  * \endenglish
00067  * @brief
00068  * @zh AGVC 自动充电、强制上下桩、电量检测等充电相关功能
00069  * @en AGVC charging functions, including auto charging, forced docking or undocking, and battery checks.
00070  */
00071
00072 /**
00073  * \chinese
00074  * @defgroup AgvInfo AgvInfo (AGV 信息模块)
```

```
00075  * \endchinese
00076  * \english
00077  * @defgroup AgvInfo AgvInfo
00078  * \endenglish
00079  * @brief
00080  * @zh AGVC 状态、位姿、日志、错误码等信息查询功能
00081  * @en Query functions for AGVC status, pose, logs, error codes, and related information.
00082  */
00083
00084 /**
00085  * \chinese
00086  * @defgroup System System (系统模块)
00087  * \endchinese
00088  * \english
00089  * @defgroup System System
00090  * \endenglish
00091  * @brief
00092  * @zh AGVC 系统配置、控制权、软件升级、固件更新等系统级功能
00093  * @en System-level functions for AGVC configuration, control priority, software upgrades, and firmware updates.
00094  */
00095
00096 namespace agvc_interface {
00097
00098 class AgvcInterface
00099 {
00100 public:
00101     AgvcInterface();
00102     virtual ~AgvcInterface();
00103
00104     /**
00105      * @ingroup System
00106      * @brief
00107      * @zh 设置 AGV 系统时间。
00108      * @en Sets the AGV system time.
00109      * @param[in] stamp
00110      * @zh 时间戳，格式为 YYYY-MM-DDTHH:mm:ss.ssZ，例如“2017-04-15T11:40:03.12Z”。
00111      * @en Timestamp in YYYY-MM-DDTHH:mm:ss.ssZ format, for example “2017-04-15T11:40:03.12Z”.
00112      */
00113     int setSystemClock(const std::string &stamp);
00114
00115     /**
00116      * @ingroup AgvInfo
00117      * @attention
00118      * @zh RTDE 推送。
00119      * @en RTDE pushed data.
00120      * @brief
00121      * @zh 查询当前 AGV 信息。
00122      * @en Queries current AGV information.
00123      * @return
00124      * @zh AGV 详细信息。
00125      * @en Detailed AGV information.
00126      */
00127     AgvDetails getAgvDetails();
00128
00129     /**
00130      * @ingroup AgvInfo
00131      * @attention
00132      * @zh RTDE 推送。
00133      * @en RTDE pushed data.
00134      * @brief
00135      * @zh 查询当前 AGV 运行信息。
00136      * @en Queries current AGV running information.
00137      * @return
00138      * @zh AGV 运行信息。
00139      * @en AGV running information.
00140      */
00141     RunningInfo getRunningInfo();
00142
00143     /**
00144      * @ingroup AgvInfo
00145      * @attention
00146      * @zh RTDE 推送。
00147      * @en RTDE pushed data.
00148      * @brief
00149      * @zh 查询 AGV 当前位姿。
00150      * @en Queries the current AGV pose.
00151      * @return
00152      * @zh 当前位姿。
00153      * @en Current pose.
00154      */
00155     Pose2d getAgvCurrentPose();
00156
00157     /**
00158      * @ingroup AgvInfo
00159      * @attention
00160      * @zh RTDE 推送。
00161      * @en RTDE pushed data.
```

```
00162      * @brief
00163      * @zh 查询当前激光点云数据。
00164      * @en Queries the current laser point cloud data.
00165      * @return
00166      * @zh 二维激光点坐标。
00167      * @en 2D laser point coordinates.
00168      */
00169     std::vector<Point2d> getLaserPointCloud();
00170
00171     /**
00172      * @ingroup System
00173      * @brief
00174      * @zh 查询异步接口的运行情况，不代表异步接口本身的运行结果。
00175      * @en Queries the running status of asynchronous APIs; this does not represent each API's own execution result.
00176      * @note
00177      * @zh 异步接口：saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv。
00178      * @en Asynchronous APIs: saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv.
00179      * @note
00180      * @zh 异步接口：getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv。
00181      * @en Asynchronous APIs: getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMapFromAgv.
00182      * @note
00183      * @zh 异步接口：getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv。
00184      * @en Asynchronous APIs: getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv.
00185      * @note
00186      * @zh 异步接口：autoAlignRailway。
00187      * @en Asynchronous API: autoAlignRailway.
00188      * @note
00189      * @zh 客户端与 AGV 断联后，可调用该接口获取异步接口运行情况。
00190      * @en After the client disconnects from the AGV, call this API to obtain asynchronous API status.
00191      * @note
00192      * @zh 返回值的每个字段中均有 header，其中 header.id 表示下发异步任务时的 id 号。
00193      * @en Each field in the return value contains a header, where header.id is the ID used when issuing the
    asynchronous task.
00194      * @return
00195      * @zh NONE：异步接口未运行；RUNNING：异步接口运行中；FINISH：异步接口运行结束。
00196      * @en NONE: asynchronous API is not running; RUNNING: asynchronous API is running; FINISH: asynchronous API has
    finished.
00197      */
00198     AsyncInterfaceResultStatus getAsyncInterfaceResultStatus();
00199
00200     /**
00201      * @ingroup Station
00202      * @brief
00203      * @zh 查询当前地图的所有站点信息。
00204      * @en Queries all station information on the current map.
00205      * @return
00206      * @zh 当前地图的所有站点信息。
00207      * @en All station information on the current map.
00208      */
00209     std::vector<StationMark> getAllStations();
00210
00211     /**
00212      * @ingroup Station
00213      * @brief
00214      * @zh 查询指定地图的所有站点信息。
00215      * @en Queries all station information on the specified map.
00216      * @param[in] map_header
00217      * @zh 指定地图 header(命令 id，名称，时间，地图 id)。
00218      * @en Header of the specified map (command ID, name, time, map ID).
00219      * @return
00220      * @zh 指定地图的所有站点信息。
00221      * @en All station information on the specified map.
00222      */
00223     std::vector<StationMark> getAllStationsOfTargetMap(const Header &map_header);
00224
00225     /**
00226      * @ingroup Station
00227      * @brief
00228      * @zh 添加或修改一个站点。
00229      * @en Adds or modifies one station.
00230      * @param[in] station
00231      * @zh 待添加或修改的站点信息。
00232      * @en Station information to add or modify.
00233      * @return
00234      * @zh 10100000：添加或修改站点成功；else：添加或修改站点失败。
00235      * @en 10100000: station added or modified successfully; else: failed to add or modify the station.
00236      */
00237     int addStation(const StationMark &station);
00238
00239     /**
00240      * @ingroup Station
00241      * @brief
00242      * @zh 添加或修改多个站点。
00243      * @en Adds or modifies multiple stations.
00244      * @param[in] stations
00245      * @zh 待添加或修改的站点信息。
00246      * @en Station information to add or modify.
```

```
00247      * @return
00248      * @zh 10100000：添加或修改站点成功；else：添加或修改站点失败。
00249      * @en 10100000: stations added or modified successfully; else: failed to add or modify stations.
00250      */
00251     int addStations(const std::vector<StationMark> &stations);
00252
00253     /**
00254      * @ingroup Station
00255      * @brief
00256      * @zh 通过自动识别充电桩位姿来添加充电站点。
00257      * @en Adds a charging station by automatically recognizing the charging board pose.
00258      * @since 0.7.0
00259      * @param[in] station_header
00260      * @zh 待添加或修改的充电站点信息（站点 id，名称，时间，地图 id）。
00261      * @en Charging station information to add or modify (station ID, name, time, map ID).
00262      * @param[in] dis_station_board
00263      * @zh 充电站点与充电桩之间的距离（充电站点位于 AGV 中心），单位：m，范围 [1.0~2.0]。
00264      * @en Distance between the charging station and charging board. The charging station is at the AGV center.
     Unit: m; range: [1.0, 2.0].
00265      * @return
00266      * @zh 10100000：添加充电站点成功；else：添加充电站点失败。
00267      * @en 10100000: charging station added successfully; else: failed to add the charging station.
00268      */
00269     int addCPStationUseChargingBoard(const Header &station_header, const double &dis_station_board = 1.5);
00270
00271     /**
00272      * @ingroup Station
00273      * @brief
00274      * @zh 删除指定站点，会删除与该站点相关的路径。
00275      * @en Deletes the specified station and its related paths.
00276      * @param[in] station_header
00277      * @zh 待删除站点信息（站点 id，名称，时间，地图 id）。
00278      * @en Station information to delete (station ID, name, time, map ID).
00279      * @return
00280      * @zh 10100000：删除站点成功；else：删除站点失败。
00281      * @en 10100000: station deleted successfully; else: failed to delete the station.
00282      */
00283     int deleteStation(const Header &station_header);
00284
00285     /**
00286      * @ingroup Station
00287      * @brief
00288      * @zh 删除多个站点，会删除与站点相关的路径。
00289      * @en Deletes multiple stations and their related paths.
00290      * @param[in] stations_header
00291      * @zh 待删除站点信息（站点 id，名称，时间，地图 id）。
00292      * @en Station information to delete (station ID, name, time, map ID).
00293      * @return
00294      * @zh 10100000：删除站点成功；else：删除站点失败。
00295      * @en 10100000: stations deleted successfully; else: failed to delete stations.
00296      */
00297     int deleteStations(const std::vector<Header> &stations_header);
00298
00299     /**
00300      * @ingroup Path
00301      * @brief
00302      * @zh 查询当前地图的所有路径信息。
00303      * @en Queries all path information on the current map.
00304      * @return
00305      * @zh 当前地图的所有路径信息。
00306      * @en All path information on the current map.
00307      */
00308     std::vector<PathStation> getAllPaths();
00309
00310     /**
00311      * @ingroup Path
00312      * @brief
00313      * @zh 查询指定地图的所有路径信息。
00314      * @en Queries all path information on the specified map.
00315      * @param[in] map_header
00316      * @zh 指定地图 header(命令 id，名称，时间，地图 id)。
00317      * @en Header of the specified map (command ID, name, time, map ID).
00318      * @return
00319      * @zh 指定地图的所有路径信息。
00320      * @en All path information on the specified map.
00321      */
00322     std::vector<PathStation> getAllPathsOfTargetMap(const Header &map_header);
00323
00324     /**
00325      * @ingroup Path
00326      * @brief
00327      * @zh 查询 AGV 当前正在跟踪的路径信息。
00328      * @en Queries the path currently being tracked by the AGV.
00329      * @return
00330      * @zh 当前正在跟踪的路径信息。
00331      * @en Information about the path currently being tracked.
00332      */
```

```
00333     PathStation getCurrentPath();
00334
00335     /**
00336      * @ingroup Path
00337      * @brief
00338      * @zh 生成或修改一条路径。
00339      * @en Generates or modifies one path.
00340      * @param[in] path_station
00341      * @zh 待生成或修改的路径信息。
00342      * @en Path information to generate or modify.
00343      * @return
00344      * @zh 10100000：修改或生成路径成功；else：修改或生成路径失败。
00345      * @en 10100000: path modified or generated successfully; else: failed to modify or generate the path.
00346      */
00347     int generatePath(const PathStation &path_station);
00348
00349     /**
00350      * @ingroup Path
00351      * @brief
00352      * @zh 生成或修改多条路径。
00353      * @en Generates or modifies multiple paths.
00354      * @param[in] paths_station
00355      * @zh 待生成或修改的路径信息。
00356      * @en Path information to generate or modify.
00357      * @return
00358      * @zh 10100000：修改或生成路径成功；else：修改或生成路径失败。
00359      * @en 10100000: paths modified or generated successfully; else: failed to modify or generate paths.
00360      */
00361     int generatePaths(const std::vector<PathStation> &paths_station);
00362
00363     /**
00364      * @ingroup Path
00365      * @brief
00366      * @zh 删除一条路径。
00367      * @en Deletes one path.
00368      * @param[in] path_header
00369      * @zh 待删除路径信息（路径 id，名称，时间，地图 id）。
00370      * @en Path information to delete (path ID, name, time, map ID).
00371      * @return
00372      * @zh 10100000：删除路径成功；else：删除路径失败。
00373      * @en 10100000: path deleted successfully; else: failed to delete the path.
00374      */
00375     int deletePath(const Header &path_header);
00376
00377     /**
00378      * @ingroup Path
00379      * @brief
00380      * @zh 删除多条路径。
00381      * @en Deletes multiple paths.
00382      * @param[in] paths_header
00383      * @zh 待删除路径信息（路径 id，名称，时间，地图 id）。
00384      * @en Path information to delete (path ID, name, time, map ID).
00385      * @return
00386      * @zh 10100000：删除路径成功；else：删除路径失败。
00387      * @en 10100000: paths deleted successfully; else: failed to delete paths.
00388      */
00389     int deletePaths(const std::vector<Header> &paths_header);
00390
00391     /**
00392      * @ingroup Map
00393      * @brief
00394      * @zh 获取 AGV 所有地图的信息头。
00395      * @en Gets the headers of all maps on the AGV.
00396      * @return
00397      * @zh 所有地图的信息头（命令 id，名称，时间，地图 id）。
00398      * @en Headers of all maps (command ID, name, time, map ID).
00399      */
00400     std::vector<Header> getMapList();
00401
00402     /**
00403      * @ingroup Map
00404      * @brief
00405      * @zh 查询当前 AGV 地图的信息头。
00406      * @en Queries the header of the current AGV map.
00407      * @return
00408      * @zh 当前地图的信息头（命令 id，名称，时间，地图 id）。
00409      * @en Header of the current map (command ID, name, time, map ID).
00410      */
00411     Header getCurrentMapHeader();
00412
00413     /**
00414      * @ingroup Map
00415      * @brief
00416      * @zh 获取指定栅格地图信息。
00417      * @en Gets the specified occupancy grid map information.
00418      * @param[in] map_header
00419      * @zh 指定地图的信息头（本次异步命令 id，名称，时间，地图 id）。
```

```
00420      * @en Header of the specified map (this asynchronous command ID, name, time, map ID).
00421      * @note
00422      * @zh map_id 为"current_map" 时，可特指为当前地图。
00423      * @en When map_id is "current_map", it refers to the current map.
00424      * @return
00425      * @zh 指定栅格地图信息。
00426      * @en Specified occupancy grid map information.
00427      */
00428     OccupancyGridMap getGridMapFromAgv(const Header &map_header);
00429
00430     /**
00431      * @ingroup Map
00432      * @attention
00433      * @zh 1. 异步接口。
00434      * @en 1. Asynchronous API.
00435      * @brief
00436      * @zh 获取 Base64 编码的 PNG 地图信息。
00437      * @en Gets Base64-encoded PNG map information.
00438      * @param[in] map_header
00439      * @zh 指定地图的信息头（本次异步命令 id，名称，时间，地图 id)。
00440      * @en Header of the specified map (this asynchronous command ID, name, time, map ID).
00441      * @note
00442      * @zh map_id 为"current_map" 时，可特指为当前地图。
00443      * @en When map_id is "current_map", it refers to the current map.
00444      * @return
00445      * @zh Base64 编码的 PNG 地图信息。
00446      * @en Base64-encoded PNG map information.
00447      */
00448     Base64PngMap getBase64PngMapFromAgv(const Header &map_header);
00449
00450     /**
00451      * @ingroup Map
00452      * @attention
00453      * @zh 1. 异步接口。
00454      * @en 1. Asynchronous API.
00455      * @brief
00456      * @zh 获取预览图，可指定预览图的长和宽。
00457      * @en Gets a preview image with optional width and height.
00458      * @param[in] map_header
00459      * @zh 指定地图的信息头（本次异步命令 id，名称，时间，地图 id)。
00460      * @en Header of the specified map (this asynchronous command ID, name, time, map ID).
00461      * @param[in] image_width_px
00462      * @zh 地图宽度（像素），默认值为 0 时返回缩略图数据为空。
00463      * @en Map width in pixels. If the default value 0 is used, returned thumbnail data is empty.
00464      * @param[in] image_height_px
00465      * @zh 地图高度（像素），默认值为 0 时返回缩略图数据为空。
00466      * @en Map height in pixels. If the default value 0 is used, returned thumbnail data is empty.
00467      * @note
00468      * @zh map_id 为"current_map" 时，可特指为当前地图。
00469      * @en When map_id is "current_map", it refers to the current map.
00470      * @return
00471      * @zh Base64 编码的 PNG 缩略图。
00472      * @en Base64-encoded PNG thumbnail.
00473      */
00474     Base64PngMap previewPngMapFromAgv(const Header &map_header, const int &image_width_px = 0, const int
      &image_height_px = 0);
00475
00476     /**
00477      * @ingroup Map
00478      * @attention
00479      * @zh 1. 异步接口。
00480      * @en 1. Asynchronous API.
00481      * @brief
00482      * @zh 将栅格地图信息发送给 AGV。
00483      * @en Sends occupancy grid map information to the AGV.
00484      * @param[in] map
00485      * @zh 栅格地图信息。
00486      * @en Occupancy grid map information.
00487      * @return
00488      * @zh 10100000：发送地图成功；10100201：发送地图中；else：发送地图失败。
00489      * @en 10100000: map sent successfully; 10100201: map is being sent; else: failed to send the map.
00490      */
00491     int sendGridMapToAgv(const OccupancyGridMap &map);
00492
00493     /**
00494      * @ingroup Map
00495      * @attention
00496      * @zh 1. 异步接口。
00497      * @en 1. Asynchronous API.
00498      * @brief
00499      * @zh 将 Base64 编码的 PNG 地图发送给 AGV。
00500      * @en Sends a Base64-encoded PNG map to the AGV.
00501      * @param[in] map
00502      * @zh Base64 编码的 PNG 地图。
00503      * @en Base64-encoded PNG map.
00504      * @return
00505      * @zh 10100000：发送地图成功；10100201：发送地图中；else：发送地图失败。
```

```
00506      * @en 10100000: map sent successfully; 10100201: map is being sent; else: failed to send the map.
00507      */
00508     int sendBase64PngMapToAgv(const Base64PngMap &map);
00509
00510     /**
00511      * @ingroup Map
00512      * @attention
00513      * @zh 1. 异步接口；2. 控制权限制。
00514      * @en 1. Asynchronous API; 2. Requires control priority.
00515      * @brief
00516      * @zh 建图完成后调用该接口保存地图。
00517      * @en Saves the map after mapping is complete.
00518      * @param[in] map_header
00519      * @zh 地图信息头（本次异步命令 id，名称，时间，地图 id)。
00520      * @en Map header (this asynchronous command ID, name, time, map ID).
00521      * @return
00522      * @zh 10100000：保存地图成功；10100201：异步接口运行中；10120202：无控制权；else：保存地图失败。
00523      * @en 10100000: map saved successfully; 10100201: asynchronous API is running; 10120202: no control priority;
      else: failed to save the map.
00524      */
00525     int saveMap(const Header &map_header);
00526
00527     /**
00528      * @ingroup Map
00529      * @attention
00530      * @zh 1. 异步接口；2. 控制权限制。
00531      * @en 1. Asynchronous API; 2. Requires control priority.
00532      * @brief
00533      * @zh 切换指定地图。
00534      * @en Switches to the specified map.
00535      * @param[in] map_header
00536      * @zh 指定地图的信息头（本次异步命令 id，名称，时间，地图 id)。
00537      * @en Header of the specified map (this asynchronous command ID, name, time, map ID).
00538      * @return
00539      * @zh 10100000：切换地图成功；10100201：异步接口运行中；10120202：无控制权；else：切换地图失败。
00540      * @en 10100000: map switched successfully; 10100201: asynchronous API is running; 10120202: no control
      priority; else: failed to switch the map.
00541      */
00542     int switchMap(const Header &map_header);
00543
00544     /**
00545      * @ingroup Map
00546      * @brief
00547      * @zh 删除一张指定地图。
00548      * @en Deletes one specified map.
00549      * @param[in] map_header
00550      * @zh 指定地图的信息头（命令 id，名称，时间，地图 id)。
00551      * @en Header of the specified map (command ID, name, time, map ID).
00552      * @return
00553      * @zh 10100000：删除地图成功；else：删除地图失败。
00554      * @en 10100000: map deleted successfully; else: failed to delete the map.
00555      */
00556     int deleteMap(const Header &map_header);
00557
00558     /**
00559      * @ingroup Map
00560      * @brief
00561      * @zh 删除多张指定地图。
00562      * @en Deletes multiple specified maps.
00563      * @param[in] map_headers
00564      * @zh 多个指定地图的信息头（命令 id，名称，时间，地图 id)。
00565      * @en Headers of the specified maps (command ID, name, time, map ID).
00566      * @return
00567      * @zh 10100000：删除地图成功；else：删除地图失败。
00568      * @en 10100000: maps deleted successfully; else: failed to delete maps.
00569      */
00570     int deleteMaps(const std::vector<Header> &map_headers);
00571
00572     /**
00573      * @ingroup Map
00574      * @brief
00575      * @zh 查询当前地图的虚拟区域。
00576      * @en Queries virtual areas on the current map.
00577      * @return
00578      * @zh 当前地图的虚拟区域信息。
00579      * @en Virtual area information on the current map.
00580      */
00581     std::vector<MapVirtualArea> getAllMapVirtualArea();
00582
00583     /**
00584      * @ingroup Map
00585      * @brief
00586      * @zh 查询目标地图的虚拟区域。
00587      * @en Queries virtual areas on the target map.
00588      * @param[in] map_header
00589      * @zh 指定地图 header(命令 id，名称，时间，地图 id)。
00590      * @en Header of the specified map (command ID, name, time, map ID).
```

```
00591      * @return
00592      * @zh 指定地图的所有虚拟区域。
00593      * @en All virtual areas on the specified map.
00594      */
00595     std::vector<MapVirtualArea> getAllMapVirtualAreaOfTargetMap(const Header &map_header);
00596
00597     /**
00598      * @ingroup Map
00599      * @brief
00600      * @zh 添加或修改一个虚拟区域。
00601      * @en Adds or modifies one virtual area.
00602      * @param[in] map_virtual_area
00603      * @zh 待添加或修改的虚拟区域信息。
00604      * @en Virtual area information to add or modify.
00605      * @return
00606      * @zh 10100000：添加或修改虚拟区域成功；else：添加或修改虚拟区域失败。
00607      * @en 10100000: virtual area added or modified successfully; else: failed to add or modify the virtual area.
00608      */
00609     int addMapVirtualArea(const MapVirtualArea &map_virtual_area);
00610
00611     /**
00612      * @ingroup Map
00613      * @brief
00614      * @zh 添加或修改多个虚拟区域。
00615      * @en Adds or modifies multiple virtual areas.
00616      * @param[in] map_virtual_areas
00617      * @zh 待添加或修改的虚拟区域信息。
00618      * @en Virtual area information to add or modify.
00619      * @return
00620      * @zh 10100000：添加或修改虚拟区域成功；else：添加或修改虚拟区域失败。
00621      * @en 10100000: virtual areas added or modified successfully; else: failed to add or modify virtual areas.
00622      */
00623     int addMapVirtualAreas(const std::vector<MapVirtualArea> &map_virtual_areas);
00624
00625     /**
00626      * @ingroup Map
00627      * @brief
00628      * @zh 删除指定虚拟区域。
00629      * @en Deletes the specified virtual area.
00630      * @param[in] virtual_area_header
00631      * @zh 指定虚拟区域信息头（命令 id，虚拟区域名称，时间，所在地图 id）。
00632      * @en Header of the specified virtual area (command ID, virtual area name, time, map ID).
00633      * @return
00634      * @zh 10100000：删除虚拟区域成功；else：删除虚拟区域失败。
00635      * @en 10100000: virtual area deleted successfully; else: failed to delete the virtual area.
00636      */
00637     int deleteMapVirtualArea(const Header &virtual_area_header);
00638
00639     /**
00640      * @ingroup Map
00641      * @brief
00642      * @zh 删除多个指定虚拟区域。
00643      * @en Deletes multiple specified virtual areas.
00644      * @param[in] virtual_areas_header
00645      * @zh 指定虚拟区域信息头（命令 id，虚拟区域名称，时间，所在地图 id）。
00646      * @en Headers of the specified virtual areas (command ID, virtual area name, time, map ID).
00647      * @return
00648      * @zh 10100000：删除虚拟区域成功；else：删除虚拟区域失败。
00649      * @en 10100000: virtual areas deleted successfully; else: failed to delete virtual areas.
00650      */
00651     int deleteMapVirtualAreas(const std::vector<Header> &virtual_areas_header);
00652
00653     /**
00654      * @ingroup Map
00655      * @attention
00656      * @zh 1. 异步接口。
00657      * @en 1. Asynchronous API.
00658      * @brief
00659      * @zh 获取指定地图的栅格地图数据、路径、站点、虚拟区信息。
00660      * @en Gets occupancy grid map data, paths, stations, and virtual areas for the specified map.
00661      * @param[in] map_header
00662      * @zh 指定地图信息头（本次异步命令 id，地图名称，时间，地图 id）。
00663      * @en Header of the specified map (this asynchronous command ID, map name, time, map ID).
00664      * @note
00665      * @zh map_id 为"current_map" 时，可特指为当前地图。
00666      * @en When map_id is "current_map", it refers to the current map.
00667      * @return
00668      * @zh 栅格地图上的全部信息。
00669      * @en All information on the occupancy grid map.
00670      */
00671     MapAllInfo getGridMapAllInfo(const Header &map_header);
00672
00673     /**
00674      * @ingroup Map
00675      * @attention
00676      * @zh 1. 异步接口。
00677      * @en 1. Asynchronous API.
```

```
00678      * @brief
00679      * @zh 获取指定地图的 Base64 地图数据、路径、站点、虚拟区信息。
00680      * @en Gets Base64 map data, paths, stations, and virtual areas for the specified map.
00681      * @param[in] map_header
00682      * @zh 指定地图信息头（本次异步命令 id，地图名称，时间，地图 id)。
00683      * @en Header of the specified map (this asynchronous command ID, map name, time, map ID).
00684      * @note
00685      * @zh map_id 为"current_map" 时，可特指为当前地图。
00686      * @en When map_id is "current_map", it refers to the current map.
00687      * @return
00688      * @zh Base64 地图上的全部信息。
00689      * @en All information on the Base64 map.
00690      */
00691     MapAllInfo getPngMapAllInfo(const Header &map_header);
00692
00693     /**
00694      * @ingroup Map
00695      * @attention
00696      * @zh 1. 异步接口；2. 控制权限制。
00697      * @en 1. Asynchronous API; 2. Requires control priority.
00698      * @brief
00699      * @zh 设置栅格地图、路径、站点、虚拟区信息。
00700      * @en Sets occupancy grid map, path, station, and virtual area information.
00701      * @param[in] map_all_info
00702      * @zh 栅格地图上的全部信息。
00703      * @en All information on the occupancy grid map.
00704      * @param[in] command_header
00705      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
00706      * @en The id field is the ID of this command; other fields have no special meaning.
00707      * @note
00708      * @zh 请保证数据下发的可行性，AGV 不会对数据进行校验。
00709      * @en Ensure that the data can be applied; the AGV does not validate the data.
00710      * @return
00711      * @zh 10100000：上传全部地图信息成功；10100201：异步接口运行中；10120202：无控制权；else：上传全部地图信息失败。
00712      * @en 10100000: all map information uploaded successfully; 10100201: asynchronous API is running; 10120202: no
     control priority; else: failed to upload all map information.
00713      */
00714     int setGridMapAllInfo(const MapAllInfo &map_all_info,
00715                           const Header &command_header = { "99999", "99999", 1, "99999" });
00716
00717     /**
00718      * @ingroup Map
00719      * @attention
00720      * @zh 1. 异步接口；2. 控制权限制。
00721      * @en 1. Asynchronous API; 2. Requires control priority.
00722      * @brief
00723      * @zh 设置 Base64 格式地图、路径、站点、虚拟区信息。
00724      * @en Sets Base64 map, path, station, and virtual area information.
00725      * @param[in] map_all_info
00726      * @zh Base64 格式地图上的全部信息。
00727      * @en All information on the Base64 map.
00728      * @param[in] command_header
00729      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
00730      * @en The id field is the ID of this command; other fields have no special meaning.
00731      * @note
00732      * @zh 请保证数据下发的可行性，AGV 不会对数据进行校验。
00733      * @en Ensure that the data can be applied; the AGV does not validate the data.
00734      * @return
00735      * @zh 10100000：上传全部地图信息成功；10100201：异步接口运行中；10120202：无控制权；else：上传全部地图信息失败。
00736      * @en 10100000: all map information uploaded successfully; 10100201: asynchronous API is running; 10120202: no
     control priority; else: failed to upload all map information.
00737      */
00738     int setPngMapAllInfo(const MapAllInfo &map_all_info,
00739                          const Header &command_header = { "99999", "99999", 1, "99999" });
00740
00741     /**
00742      * @ingroup Map
00743      * @brief
00744      * @zh 删除指定地图的全部信息，包括地图数据、站点、路径、虚拟区信息。
00745      * @en Deletes all information of the specified map, including map data, stations, paths, and virtual areas.
00746      * @param[in] map_header
00747      * @zh 指定地图信息头（命令 id，地图名称，时间，地图 id)。
00748      * @en Header of the specified map (command ID, map name, time, map ID).
00749      * @return
00750      * @zh 10100000：删除成功；else：删除失败。
00751      * @en 10100000: deleted successfully; else: deletion failed.
00752      */
00753     int deleteMapAllInfo(const Header &map_header);
00754
00755     /**
00756      * @ingroup System
00757      * @attention
00758      * @zh 异步接口。
00759      * @en Asynchronous API.
00760      * @brief
00761      * @zh 获取当前扫描到的 WiFi 列表。
00762      * @en Gets the currently scanned WiFi list.
```

```
00763      * @return
00764      * @zh {RUNNING}: 异步接口运行中; {}: 为空，未扫描到 WIFI; {SSID}: 不为空，扫描到的 WIFI 名称列表。
00765      * @en {RUNNING}: asynchronous API is running; {}: empty, no WiFi found; {SSID}: non-empty list of scanned WiFi
     SSIDs.
00766      */
00767     std::vector<std::string> getWifiList();
00768
00769     /**
00770      * @ingroup System
00771      * @brief
00772      * @zh 连接到指定 WiFi，自动切换到 WiFi 模式。
00773      * @en Connects to the specified WiFi and automatically switches to WiFi mode.
00774      * @note
00775      * @zh 必须在有线连接时调用，且执行时间较长。
00776      * @en Must be called over a wired connection and may take a long time.
00777      * @param[in] ssid
00778      * @zh WiFi 名称。
00779      * @en WiFi SSID.
00780      * @param[in] password
00781      * @zh WiFi 密码。
00782      * @en WiFi password.
00783      * @return
00784      * @zh 10100000: 连接 WiFi 成功; else: 连接 WiFi 失败。
00785      * @en 10100000: connected to WiFi successfully; else: failed to connect to WiFi.
00786      */
00787     int connectWifi(const std::string &ssid, const std::string &password);
00788
00789     /**
00790      * @ingroup System
00791      * @brief
00792      * @zh 开启热点，自动切换到热点模式。
00793      * @en Enables hotspot mode and automatically switches to hotspot mode.
00794      * @note
00795      * @zh 必须在有线连接时调用，且执行时间较长。
00796      * @en Must be called over a wired connection and may take a long time.
00797      * @param[in] password
00798      * @zh 热点密码; 如果为空使用"administrator" 为默认密码，SSID 默认为 SN 码。
00799      * @en Hotspot password. If empty, "administrator" is used as the default password, and the SSID defaults to the
     SN.
00800      * @return
00801      * @zh 10100000: 启用热点成功; else: 启用热点失败。
00802      * @en 10100000: hotspot enabled successfully; else: failed to enable hotspot.
00803      */
00804     int enableHotspot(const std::string &password);
00805
00806     /**
00807      * @ingroup System
00808      * @brief
00809      * @zh 获取本机所有 IP 地址。
00810      * @en Gets all local IP addresses.
00811      * @return
00812      * @zh 本机所有 IP 地址。
00813      * @en All local IP addresses.
00814      */
00815     std::vector<IpAddressInfo> getIpAddressList();
00816
00817     /**
00818      * @ingroup Navigation
00819      * @attention
00820      * @zh 1. 异步接口; 2. 控制权限制。
00821      * @en 1. Asynchronous API; 2. Requires control priority.
00822      * @brief
00823      * @zh 切换 AGV 模式。
00824      * @en Switches the AGV running mode.
00825      * @param[in] running_mode
00826      * @zh 打算切换的模式，如导航模式、建图模式。
00827      * @en Target mode to switch to, such as navigation mode or mapping mode.
00828      * @param[in] command_header
00829      * @zh id 字段代表下发本次命令的 id; 其他字段无特殊含义。
00830      * @en The id field is the ID of this command; other fields have no special meaning.
00831      * @return
00832      * @zh 10100000: 切换模式成功; 10100201: 异步接口运行中; 10120202: 无控制权; else: 切换模式失败。
00833      * @en 10100000: mode switched successfully; 10100201: asynchronous API is running; 10120202: no control
     priority; else: failed to switch mode.
00834      */
00835     int changeRunningMode(const RunningMode &running_mode,
00836                           const Header &command_header = { "99999", "99999", 1, "99999" });
00837
00838     /**
00839      * @ingroup Navigation
00840      * @attention
00841      * @zh 控制权限制。
00842      * @en Requires control priority.
00843      * @brief
00844      * @zh 控制 AGV 运动。
00845      * @en Controls AGV motion.
00846      * @param[in] speed
```

```
00847      * @zh 速度信息。
00848      * @en Speed information.
00849      * @return
00850      * @zh 10100000：向 AGV 下发速度成功；10120202：没有控制权；else：向 AGV 下发速度失败。
00851      * @en 10100000: speed command sent to the AGV successfully; 10120202: no control priority; else: failed to send
       the speed command.
00852      */
00853     int setControlSpeed(const Speed &speed);
00854
00855     /**
00856      * @ingroup Navigation
00857      * @attention
00858      * @zh 控制权限制。
00859      * @en Requires control priority.
00860      * @brief
00861      * @zh 向 AGV 发送导航任务。
00862      * @en Sends a navigation goal to the AGV.
00863      * @param[in] target
00864      * @zh 目标位置信息与导航参数信息。
00865      * @en Target pose information and navigation parameters.
00866      * @return
00867      * @zh 10100000：设置导航任务成功；10120202：没有控制权；else：设置导航任务失败。
00868      * @en 10100000: navigation goal set successfully; 10120202: no control priority; else: failed to set the
       navigation goal.
00869      */
00870     int setNavGoal(const NavGoalType &target);
00871
00872     /**
00873      * @ingroup AgvInfo
00874      * @attention
00875      * @zh RTDE 推送。
00876      * @en RTDE pushed data.
00877      * @brief
00878      * @zh 获取当前导航信息。
00879      * @en Gets current navigation information.
00880      * @return
00881      * @zh 当前导航信息。
00882      * @en Current navigation information.
00883      */
00884     NavInfo getNavInfo();
00885
00886     /**
00887      * @ingroup Navigation
00888      * @attention
00889      * @zh 控制权限制。
00890      * @en Requires control priority.
00891      * @brief
00892      * @zh 暂停 AGV 速度。
00893      * @en Pauses AGV speed.
00894      * @return
00895      * @zh 10100000：暂停成功；10120202：没有控制权；else：暂停失败。
00896      * @en 10100000: paused successfully; 10120202: no control priority; else: pause failed.
00897      */
00898     int pauseAgvSpeed();
00899
00900     /**
00901      * @ingroup Navigation
00902      * @attention
00903      * @zh 控制权限制。
00904      * @en Requires control priority.
00905      * @brief
00906      * @zh 暂停后恢复 AGV 速度。
00907      * @en Resumes AGV speed after a pause.
00908      * @return
00909      * @zh 10100000：恢复成功；10120202：没有控制权；else：恢复失败。
00910      * @en 10100000: resumed successfully; 10120202: no control priority; else: resume failed.
00911      */
00912     int resumeAgvSpeed();
00913
00914     /**
00915      * @ingroup Navigation
00916      * @attention
00917      * @zh 控制权限制。
00918      * @en Requires control priority.
00919      * @brief
00920      * @zh 取消导航任务。
00921      * @en Cancels the navigation task.
00922      * @return
00923      * @zh 10100000：取消导航任务成功；10120202：没有控制权；else：取消导航任务失败。
00924      * @en 10100000: navigation task canceled successfully; 10120202: no control priority; else: failed to cancel
       the navigation task.
00925      */
00926     int cancelNavigation();
00927
00928     /**
00929      * @ingroup Navigation
00930      * @attention
```

```
00931      * @zh 1. 异步接口；2. 控制权限制。
00932      * @en 1. Asynchronous API; 2. Requires control priority.
00933      * @brief
00934      * @zh AGV 重定位。
00935      * @en Relocates the AGV.
00936      * @param[in] init_pose
00937      * @zh AGV 当前参考位姿。
00938      * @en Current reference pose of the AGV.
00939      * @param[in] command_header
00940      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
00941      * @en The id field is the ID of this command; other fields have no special meaning.
00942      * @return
00943      * @zh 10100000：重定位成功；10100201：异步接口运行中；10120202：无控制权；else：重定位失败。
00944      * @en 10100000: relocation succeeded; 10100201: asynchronous API is running; 10120202: no control priority;
      else: relocation failed.
00945      */
00946     int relocation(const Pose2d &init_pose, const Header &command_header = { "99999", "99999", 1, "99999" });
00947
00948     /**
00949      * @ingroup System
00950      * @brief
00951      * @zh 获取 AGV 控制器的参数文件。
00952      * @en Gets the AGV controller parameter file.
00953      * @return
00954      * @zh 当前 AGV 控制器中的参数信息。
00955      * @en Parameter information in the current AGV controller.
00956      */
00957     std::string getAgvControllerParametersFile();
00958
00959     /**
00960      * @ingroup System
00961      * @attention
00962      * @zh 控制权限制。
00963      * @en Requires control priority.
00964      * @brief
00965      * @zh 设置 AGV 控制器的参数文件。
00966      * @en Sets the AGV controller parameter file.
00967      * @note
00968      * @zh 不要高频调用，调用频率低于 1Hz；此接口会默认执行刷新参数功能。
00969      * @en Do not call frequently. Keep the call rate below 1 Hz. This API refreshes parameters by default.
00970      * @param[in] agv_parameters
00971      * @zh AGV 控制器的参数信息。
00972      * @en Parameter information for the AGV controller.
00973      * @return
00974      * @zh 10100000：设置参数成功；else：设置参数失败。
00975      * @en 10100000: parameters set successfully; else: failed to set parameters.
00976      */
00977     int setAgvControllerParametersFile(const std::string &agv_parameters);
00978
00979     /**
00980      * @ingroup System
00981      * @attention
00982      * @zh 控制权限制。
00983      * @en Requires control priority.
00984      * @brief
00985      * @zh 修改 AGV 控制器的参数文件后，让 AGV 各节点刷新参数。
00986      * @en Refreshes parameters on AGV nodes after the AGV controller parameter file is modified.
00987      * @note
00988      * @zh 不要高频调用，调用频率低于 1Hz。
00989      * @en Do not call frequently. Keep the call rate below 1 Hz.
00990      * @return
00991      * @zh 10100000：接口执行成功；else：接口执行失败。
00992      * @en 10100000: API executed successfully; else: API execution failed.
00993      */
00994     int refreshAgvControllerParametersFile();
00995
00996     /**
00997      * @ingroup System
00998      * @attention
00999      * @zh 控制权限制。
01000      * @en Requires control priority.
01001      * @brief
01002      * @zh 将 AGV 控制器的参数文件恢复出厂设置。
01003      * @en Restores the AGV controller parameter file to factory defaults.
01004      * @note
01005      * @zh 不要高频调用，调用频率低于 1Hz。
01006      * @en Do not call frequently. Keep the call rate below 1 Hz.
01007      * @return
01008      * @zh 10100000：接口执行成功；else：接口执行失败。
01009      * @en 10100000: API executed successfully; else: API execution failed.
01010      */
01011     int resetAgvControllerParametersFile();
01012
01013     /**
01014      * @ingroup Charging
01015      * @attention
01016      * @zh 控制权限制。
```

```
01017      * @en Requires control priority.
01018      * @brief
01019      * @zh 检测当前电量，电量低将自动上桩充电。
01020      * @en Checks the current battery level and automatically docks for charging when the battery is low.
01021      * @deprecated @if Chinese 自 0.7.0 起废弃，请使用新的 sendAutoChargingCommand(const AutoChargingCommand&) 接口。
    @endif @if English Deprecated since 0.7.0. Use sendAutoChargingCommand(const AutoChargingCommand&) instead. @endif
01022      * @note
01023      * @zh 充电完成后，根据配置文件中的参数 auto_leave_board，决定是否自动下桩并行驶到指定站点或等待区。
01024      * @en After charging completes, the auto_leave_board parameter decides whether the AGV automatically leaves the
    board and moves to a target station or waiting area.
01025      * @note
01026      * @zh 在配置文件中设置电量低阈值 [low_battery_threshold] 或电量高阈值 [high_battery_threshold]。
01027      * @en Configure the low battery threshold [low_battery_threshold] or high battery threshold
    [high_battery_threshold] in the configuration file.
01028      * @param[in] check_power_header
01029      * @zh 本次命令的 id，本次命令的 name，当前时间，地图 id。
01030      * @en ID of this command, name of this command, current time, and map ID.
01031      * @param[in] nav_board_charge_station_id
01032      * @zh 行驶到指定的充电站点进行上桩充电；如果不设置该参数，行驶到最近的充电站点进行上桩充电。
01033      * @en Drives to the specified charging station for docking and charging. If unset, drives to the nearest
    charging station.
01034      * @return
01035      * @zh 10100000：接口调用成功，AGV 开始执行充电命令；10120202：无控制权；else：接口调用失败。
01036      * @en 10100000: API call succeeded and the AGV starts the charging command; 10120202: no control priority;
    else: API call failed.
01037      */
01038     [[deprecated("Since 0.7.0: use sendAutoChargingCommand(const AutoChargingCommand&) instead.")]]
01039     int checkPowerAndAutoCharge(const Header &check_power_header               = { "99999", "99999", 99999, "99999"
    },
01040                                 const std::string &nav_board_charge_station_id = "99999");
01041
01042     /**
01043      * @ingroup Charging
01044      * @brief
01045      * @zh 设置自动下桩时的目标站点。
01046      * @en Sets the target station for automatic undocking.
01047      * @deprecated @if Chinese 自 0.7.0 起废弃，请使用新的 sendAutoChargingCommand(const AutoChargingCommand&) 接口。
    @endif @if English Deprecated since 0.7.0. Use sendAutoChargingCommand(const AutoChargingCommand&) instead. @endif
01048      * @param[in] leave_board_target_station_header
01049      * @zh 自动下桩的站点 header 信息。
01050      * @en Header information of the station used for automatic undocking.
01051      * @return
01052      * @zh 10100000：设置自动下桩站点成功；else：设置自动下桩站点失败。
01053      * @en 10100000: automatic undocking station set successfully; else: failed to set the station.
01054      */
01055     [[deprecated("Since 0.7.0: use sendAutoChargingCommand(const AutoChargingCommand&) instead.")]]
01056     int setLeaveBoardTargetStation(const Header &leave_board_target_station_header);
01057
01058     /**
01059      * @ingroup Charging
01060      * @brief
01061      * @zh 获取自动下桩时的目标站点。
01062      * @en Gets the target station for automatic undocking.
01063      * @deprecated @if Chinese 自 0.7.0 起废弃，请使用新的 getNavInfo() 接口。 @endif @if English Deprecated since
    0.7.0. Use getNavInfo() instead. @endif
01064      * @return
01065      * @zh 自动下桩时的目标站点 Header 信息。
01066      * @en Header information of the target station for automatic undocking.
01067      */
01068     [[deprecated("Since 0.7.0: use getNavInfo() instead.")]]
01069     Header getLeaveBoardTargetStation();
01070
01071     /**
01072      * @ingroup Charging
01073      * @attention
01074      * @zh 控制权限制。
01075      * @en Requires control priority.
01076      * @brief
01077      * @zh 强制上桩充电。
01078      * @en Forces the AGV to dock for charging.
01079      * @deprecated @if Chinese 自 0.7.0 起废弃，请使用新的 sendAutoChargingCommand(const AutoChargingCommand&) 接口。
    @endif @if English Deprecated since 0.7.0. Use sendAutoChargingCommand(const AutoChargingCommand&) instead. @endif
01080      * @param[in] forced_charge_header
01081      * @zh 本次命令的 id，本次命令的 name，当前时间，地图 id。
01082      * @en ID of this command, name of this command, current time, and map ID.
01083      * @param[in] nav_board_charge_station_id
01084      * @zh 行驶到指定的充电站点进行上桩充电；如果不设置该参数，行驶到最近的充电站点进行上桩充电。
01085      * @en Drives to the specified charging station for docking and charging. If unset, drives to the nearest
    charging station.
01086      * @return
01087      * @zh 10100000：接口调用成功，AGV 开始上桩充电；10120202：无控制权；else：接口调用失败。
01088      * @en 10100000: API call succeeded and the AGV starts docking for charging; 10120202: no control priority;
    else: API call failed.
01089      */
01090     [[deprecated("Since 0.7.0: use sendAutoChargingCommand(const AutoChargingCommand&) instead.")]]
01091     int forcedAgvToChargingBoard(const Header &forced_charge_header            = { "99999", "99999", 99999, "99999"
    },
```

```
01092                                   const std::string &nav_board_charge_station_id = "99999");
01093
01094     /**
01095      * @ingroup Charging
01096      * @attention
01097      * @zh 控制权限制。
01098      * @en Requires control priority.
01099      * @brief
01100      * @zh 强制下桩。
01101      * @en Forces the AGV to leave the charging board.
01102      * @deprecated @if Chinese 自 0.7.0 起废弃，请使用新的 sendAutoChargingCommand(const AutoChargingCommand&) 接口。
      @endif @if English Deprecated since 0.7.0. Use sendAutoChargingCommand(const AutoChargingCommand&) instead. @endif
01103      * @param[in] forced_leave_header
01104      * @zh 本次命令的 id，本次命令的 name，当前时间，地图 id。
01105      * @en ID of this command, name of this command, current time, and map ID.
01106      * @param[in] leave_board_target_station_id
01107      * @zh 下桩前往的站点；如果不设置该参数，则下桩到充电站点。
01108      * @en Station to move to after leaving the board. If unset, the AGV leaves the board to the charging station.
01109      * @return
01110      * @zh 10100000：接口调用成功，AGV 开始下桩；10120202：无控制权；else：接口调用失败。
01111      * @en 10100000: API call succeeded and the AGV starts leaving the board; 10120202: no control priority; else:
      API call failed.
01112      */
01113     [[deprecated("Since 0.7.0: use sendAutoChargingCommand(const AutoChargingCommand&) instead.")]]
01114     int forcedAgvLeaveChargingBoard(const Header &forced_leave_header = { "99999", "99999", 99999, "99999" },
01115                                     const std::string &leave_board_target_station_id = "99999");
01116
01117     /**
01118      * @ingroup Charging
01119      * @attention
01120      * @zh 控制权限制。
01121      * @en Requires control priority.
01122      * @brief
01123      * @zh 下发自动充电命令。
01124      * @en Sends an automatic charging command.
01125      * @since 0.7.0
01126      * @param[in] auto_charging_command
01127      * @zh 自动充电命令。
01128      * @en Automatic charging command.
01129      * @return
01130      * @zh 10100000：接口调用成功，AGV 开始执行充电命令；10120202：无控制权；else：接口调用失败。
01131      * @en 10100000: API call succeeded and the AGV starts the charging command; 10120202: no control priority;
      else: API call failed.
01132      */
01133     int sendAutoChargingCommand(const AutoChargingCommand &auto_charging_command);
01134
01135     /**
01136      * @ingroup AgvInfo
01137      * @brief
01138      * @zh 获取 AGV 运行中的日志信息。
01139      * @en Gets AGV runtime log messages.
01140      * @return
01141      * @zh AGV 运行中的日志信息。
01142      * @en AGV runtime log messages.
01143      */
01144     std::string getAgvLogMessage();
01145
01146     /**
01147      * @ingroup System
01148      * @brief
01149      * @zh AGV 软件升级。
01150      * @en Upgrades AGV software.
01151      * @param[in] upgrade_pack_path
01152      * @zh 升级包路径，例如"/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz"。
01153      * @en Upgrade package path, for example "/root/agvc_release-0.1.1+398d1f7-Linux_x86_64.tar.gz".
01154      * @return
01155      * @zh 10100000：目标版本升级成功；10100201：正在升级中；else：升级失败。
01156      * @en 10100000: target version upgraded successfully; 10100201: upgrade is in progress; else: upgrade failed.
01157      */
01158     int updateSoftware(const std::string &upgrade_pack_path);
01159
01160     /**
01161      * @ingroup System
01162      * @brief
01163      * @zh 获取 AGV 软件版本列表。
01164      * @en Gets the AGV software version list.
01165      * @return
01166      * @zh 软件版本列表集合 {0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}。
01167      * @en Software version list, for example {0.3.2+3b807d1-Linux_x86_64,0.6.3-patch.3+b8c6aa4-Linux_x86_64}.
01168      */
01169     std::vector<std::string> getSoftwareVersionList();
01170
01171     /**
01172      * @ingroup System
01173      * @brief
01174      * @zh 切换 AGV 软件版本。
01175      * @en Switches the AGV software version.
```

```
01176      * @param[in] software_version
01177      * @zh 软件版本，例如"0.3.2+3b807d1-Linux_x86_64"。
01178      * @en Software version, for example "0.3.2+3b807d1-Linux_x86_64".
01179      * @return
01180      * @zh 10100000：软件版本切换成功；else：软件版本切换失败。
01181      * @en 10100000: software version switched successfully; else: failed to switch software version.
01182      */
01183     int switchSoftwareVersion(const std::string &software_version);
01184
01185     /**
01186      * @ingroup System
01187      * @brief
01188      * @zh 卸载 AGV 软件版本。
01189      * @en Uninstalls an AGV software version.
01190      * @param[in] software_version
01191      * @zh 软件版本，例如"0.3.2+3b807d1-Linux_x86_64"。
01192      * @en Software version, for example "0.3.2+3b807d1-Linux_x86_64".
01193      * @return
01194      * @zh 10100000：软件版本卸载成功；else：软件版本卸载失败。
01195      * @en 10100000: software version uninstalled successfully; else: failed to uninstall software version.
01196      */
01197     int uninstallSoftware(const std::string &software_version);
01198
01199     /**
01200      * @ingroup System
01201      * @brief
01202      * @zh AGV 固件更新。
01203      * @en Updates AGV firmware.
01204      * @param[in] update_firmware
01205      * @zh 配置要更新固件的模块、固件存储路径。
01206      * @en Configures the firmware modules to update and the firmware storage path.
01207      * @return
01208      * @zh 10100000：下发固件更新信息成功；else：固件正在更新中。
01209      * @en 10100000: firmware update information sent successfully; else: firmware update is in progress.
01210      */
01211     int updateFirmware(const FirmwareUpdateParam &update_firmware);
01212
01213     /**
01214      * @ingroup System
01215      * @brief
01216      * @zh 获取固件更新过程信息。
01217      * @en Gets firmware update process information.
01218      * @return
01219      * @zh 固件更新步骤信息（步骤信息为 failed 代表更新失败），更新进度（进度信息为 1.0 代表更新成功）。
01220      * @en Firmware update step information (failed means the update failed) and update progress (1.0 means the
       update succeeded).
01221      */
01222     FirmwareUpdateProcessInfo getUpdateFirmwareProcess();
01223
01224     /**
01225      * @ingroup System
01226      * @brief
01227      * @zh 开始采集数据。
01228      * @en Starts collecting calibration data.
01229      * @note
01230      * @zh LASER_ODOM 标定采集数据量要求大于 1000 组，具体数据量可以在配置文件中修改。
01231      * @en LASER_ODOM calibration requires more than 1000 data samples. The exact amount can be changed in the
       configuration file.
01232      * @param[in] type
01233      * @zh 标定类型。
01234      * @en Calibration type.
01235      * @param[in] command_header
01236      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
01237      * @en The id field is the ID of this command; other fields have no special meaning.
01238      * @return
01239      * @zh 10100000：接口调用成功；10120202：无控制权；else：接口调用失败。
01240      * @en 10100000: API call succeeded; 10120202: no control priority; else: API call failed.
01241      */
01242     int startCollectCalibrationData(const CalibrationType &type,
01243                                     const Header &command_header = { "99999", "99999", 1, "99999" });
01244
01245     /**
01246      * @ingroup System
01247      * @brief
01248      * @zh 取消采集数据。
01249      * @en Cancels calibration data collection.
01250      * @param[in] type
01251      * @zh 标定类型。
01252      * @en Calibration type.
01253      * @param[in] command_header
01254      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
01255      * @en The id field is the ID of this command; other fields have no special meaning.
01256      * @return
01257      * @zh 10100000：接口调用成功；10120202：无控制权；else：接口调用失败。
01258      * @en 10100000: API call succeeded; 10120202: no control priority; else: API call failed.
01259      */
01260     int cancelCollectCalibrationData(const CalibrationType &type,
```

```
01261                                       const Header &command_header = { "99999", "99999", 1, "99999" });
01262
01263     /**
01264      * @ingroup System
01265      * @brief
01266      * @zh 开始标定。
01267      * @en Starts calibration.
01268      * @param[in] type
01269      * @zh 标定类型。
01270      * @en Calibration type.
01271      * @param[in] command_header
01272      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
01273      * @en The id field is the ID of this command; other fields have no special meaning.
01274      * @return
01275      * @zh 10100000：接口调用成功；10120202：无控制权；else：接口调用失败。
01276      * @en 10100000: API call succeeded; 10120202: no control priority; else: API call failed.
01277      */
01278     int startCalibration(const CalibrationType &type, const Header &command_header = { "99999", "99999", 1, "99999"
      });
01279
01280     /**
01281      * @ingroup System
01282      * @brief
01283      * @zh 获取标定信息。
01284      * @en Gets calibration information.
01285      * @param[in] type
01286      * @zh 标定类型。
01287      * @en Calibration type.
01288      * @return
01289      * @zh 标定信息。
01290      * @en Calibration information.
01291      */
01292     CalibrationProcessInfo getCalibrationProcessInfo(const CalibrationType &type);
01293
01294     /**
01295      * @ingroup System
01296      * @brief
01297      * @zh 重启 AGV。
01298      * @en Restarts the AGV.
01299      * @return
01300      * @zh 10100000：重启指令执行成功；else：重启指令执行失败。
01301      * @en 10100000: restart command executed successfully; else: restart command failed.
01302      */
01303     int restartAgv();
01304
01305     /**
01306      * @ingroup System
01307      * @brief
01308      * @zh 设置 AGV 的名称。
01309      * @en Sets the AGV name.
01310      * @param[in] agv_name
01311      * @zh 设置的 AGV 名称。
01312      * @en AGV name to set.
01313      * @return
01314      * @zh 10100000：设置名称成功；else：设置名称失败。
01315      * @en 10100000: name set successfully; else: failed to set the name.
01316      */
01317     int setAgvName(const std::string &agv_name);
01318
01319     /**
01320      * @ingroup System
01321      * @brief
01322      * @zh 设置控制权。
01323      * @en Sets control priority.
01324      * @note
01325      * @zh 1. 该接口设置成功后才能正常使用控制权相关接口；2. 控制权设置前需要先释放他人的控制权：releasePriority；3.Web
      端抢占优先权：name="web"。
01326      * @en 1. Control-priority APIs can be used only after this API succeeds; 2. Release another client's control
      priority with releasePriority before setting priority; 3. To preempt priority from the web side, set name="web".
01327      * @param[in] name
01328      * @zh 登录 RPC 注册时的用户名。
01329      * @en User name registered during RPC login.
01330      * @param[in] ip
01331      * @zh IP 地址。
01332      * @en IP address.
01333      * @return
01334      * @zh 10100000：设置控制权成功；else：设置控制权失败。
01335      * @en 10100000: control priority set successfully; else: failed to set control priority.
01336      */
01337     int setPriority(const std::string &name, const std::string &ip = "");
01338
01339     /**
01340      * @ingroup System
01341      * @brief
01342      * @zh 释放控制权。
01343      * @en Releases control priority.
01344      * @return
```

```
01345      * @zh 10100000: 释放控制权成功; else: 释放控制权失败。
01346      * @en 10100000: control priority released successfully; else: failed to release control priority.
01347      */
01348     int releasePriority();
01349
01350     /**
01351      * @ingroup System
01352      * @brief
01353      * @zh 暂停脚本。
01354      * @en Pauses the script.
01355      * @return
01356      * @zh 10100000: 脚本暂停成功; else: 脚本暂停失败。
01357      * @en 10100000: script paused successfully; else: failed to pause the script.
01358      */
01359     int scriptPaused();
01360
01361     /**
01362      * @ingroup System
01363      * @brief
01364      * @zh 恢复脚本。
01365      * @en Resumes the script.
01366      * @return
01367      * @zh 10100000: 脚本恢复成功; else: 脚本恢复失败。
01368      * @en 10100000: script resumed successfully; else: failed to resume the script.
01369      */
01370     int scriptResume();
01371
01372     /**
01373      * @ingroup System
01374      * @brief
01375      * @zh 停止脚本。
01376      * @en Stops the script.
01377      * @return
01378      * @zh 10100000: 脚本停止成功; else: 脚本停止失败。
01379      * @en 10100000: script stopped successfully; else: failed to stop the script.
01380      */
01381     int scriptStop();
01382
01383     /**
01384      * @ingroup AgvInfo
01385      * @brief
01386      * @zh 获取脚本运行状态。
01387      * @en Gets the script runtime state.
01388      * @return
01389      * @zh 脚本运行状态。
01390      * @en Script runtime state.
01391      */
01392     ScriptRuntimeState getScriptStatus();
01393
01394     /**
01395      * @ingroup System
01396      * @brief
01397      * @zh 设置脚本运行状态。
01398      * @en Sets the script runtime state.
01399      * @return
01400      * @zh 10100000: 设置脚本运行状态成功; else: 设置脚本运行状态失败。
01401      * @en 10100000: script runtime state set successfully; else: failed to set script runtime state.
01402      */
01403     int setScriptStatus(ScriptRuntimeState status);
01404
01405     /**
01406      * @ingroup AgvInfo
01407      * @brief
01408      * @zh 错误码查询。
01409      * @en Queries an error code.
01410      * @param[in] error_code
01411      * @zh 错误码。
01412      * @en Error code.
01413      * @return
01414      * @zh 错误码含义; 为空表示错误码未知。
01415      * @en Meaning of the error code. Empty means the error code is unknown.
01416      */
01417     std::string errorCodeDecoder(const int &error_code);
01418
01419     /**
01420      * @ingroup AgvInfo
01421      * @brief
01422      * @zh 获取当前的错误码。
01423      * @en Gets current error codes.
01424      * @return
01425      * @zh 错误码集合。
01426      * @en Error code collection.
01427      */
01428     std::vector<Header> getCurrentErrorCodes();
01429
01430     /**
01431      * @ingroup AgvInfo
```

```
01432      * @brief
01433      * @zh 寻找 AGV，自动播放语音并特殊灯光闪烁。
01434      * @en Locates the AGV by automatically playing audio and flashing special lights.
01435      * @return
01436      * @zh 10100000：寻找指令下发成功；else：下发指令失败。
01437      * @en 10100000: locate command sent successfully; else: failed to send the command.
01438      */
01439     int soundLightPrompt();
01440
01441     /**
01442      * @ingroup AgvInfo
01443      * @brief
01444      * @zh AGV 行进方向是否有障碍物。
01445      * @en Checks whether there is an obstacle in the AGV travel direction.
01446      * @param[in] detect_distance
01447      * @zh 检测距离，默认为 1.0m。
01448      * @en Detection distance. Default: 1.0 m.
01449      * @return
01450      * @zh true：行进方向存在障碍物；false：行进方向无障碍物。
01451      * @en true: obstacle exists in the travel direction; false: no obstacle in the travel direction.
01452      */
01453     bool isObstacleAhead(const double &detect_distance = 1.0);
01454
01455     /**
01456      * @ingroup AgvInfo
01457      * @brief
01458      * @zh 获取指定检测距离范围内的最近站点信息。
01459      * @en Gets the nearest station within the specified detection distance.
01460      * @param[in] detect_distance
01461      * @zh 检测距离，默认为 1.0m。
01462      * @en Detection distance. Default: 1.0 m.
01463      * @return
01464      * @zh 最近站点信息，id 为 NONE 代表检测距离内无站点。
01465      * @en Nearest station information. id NONE means no station is within the detection distance.
01466      */
01467     StationMark getNearestStation(const double &detect_distance = 1.0);
01468
01469     /**
01470      * @ingroup System
01471      * @attention
01472      * @zh 1. 异步接口；2. 控制权限制。
01473      * @en 1. Asynchronous API; 2. Requires control priority.
01474      * @brief
01475      * @zh 自动对接轨道并上轨。
01476      * @en Automatically aligns with the rail and boards the rail.
01477      * @param[in] command_header
01478      * @zh id 字段代表下发本次命令的 id；其他字段无特殊含义。
01479      * @en The id field is the ID of this command; other fields have no special meaning.
01480      * @return
01481      * @zh 10100000：自动上轨成功；10100201：正在上轨中；10120202：无控制权；else：上轨失败。
01482      * @en 10100000: rail boarding succeeded; 10100201: rail boarding is in progress; 10120202: no control priority;
     else: rail boarding failed.
01483      */
01484     int autoAlignRailway(const Header &command_header = { "99999", "99999", 1, "99999" });
01485
01486     /** \cond */
01487 protected:
01488     void *d_;
01489     /** \endcond */
01490 };
01491
01492 using AgvcInterfacePtr = std::shared_ptr<AgvcInterface>;
01493
01494 } // namespace agvc_interface
01495
01496 #define AgvcInterface_DECLARES                                                                    \
01497     _FUNC(AgvcInterface, 1, setSystemClock, stamp)                                                \
01498     _FUNC(AgvcInterface, 0, getAgvDetails)                                                        \
01499     _FUNC(AgvcInterface, 0, getRunningInfo)                                                       \
01500     _FUNC(AgvcInterface, 0, getAgvCurrentPose)                                                    \
01501     _FUNC(AgvcInterface, 0, getAsyncInterfaceResultStatus)                                        \
01502     _FUNC(AgvcInterface, 0, getLaserPointCloud)                                                   \
01503     _FUNC(AgvcInterface, 0, getAllStations)                                                       \
01504     _FUNC(AgvcInterface, 1, getAllStationsOfTargetMap, map_header)                                \
01505     _FUNC(AgvcInterface, 1, addStation, station)                                                  \
01506     _FUNC(AgvcInterface, 1, addStations, stations)                                                \
01507     _FUNC(AgvcInterface, 2, addCPStationUseChargingBoard, station_header, dis_station_post)       \
01508     _FUNC(AgvcInterface, 1, deleteStation, station_header)                                        \
01509     _FUNC(AgvcInterface, 1, deleteStations, stations_header)                                      \
01510     _FUNC(AgvcInterface, 0, getAllPaths)                                                          \
01511     _FUNC(AgvcInterface, 1, getAllPathsOfTargetMap, map_header)                                   \
01512     _FUNC(AgvcInterface, 0, getCurrentPath)                                                       \
01513     _FUNC(AgvcInterface, 1, generatePath, path_station)                                           \
01514     _FUNC(AgvcInterface, 1, generatePaths, paths_station)                                         \
01515     _FUNC(AgvcInterface, 1, deletePath, path_header)                                              \
01516     _FUNC(AgvcInterface, 1, deletePaths, paths_header)                                            \
01517     _FUNC(AgvcInterface, 0, getMapList)                                                           \
```

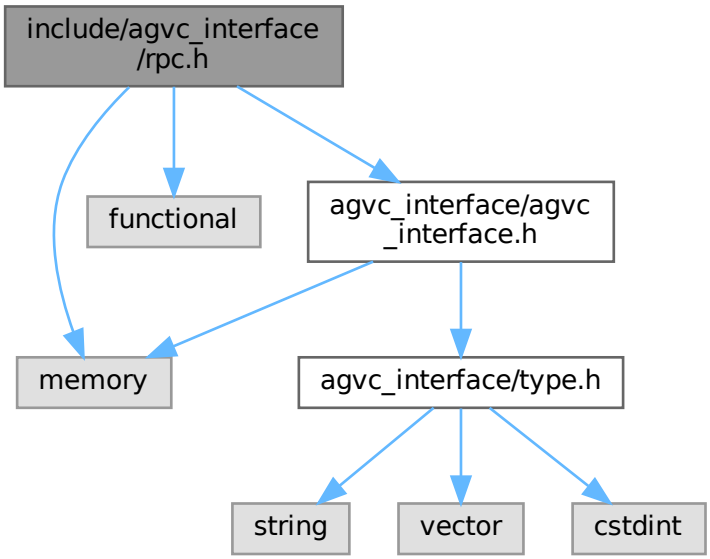
```
01518     _FUNC(AgvcInterface, 0, getCurrentMapHeader)                                                    \
01519     _FUNC(AgvcInterface, 1, getGridMapFromAgv, map_header)                                          \
01520     _FUNC(AgvcInterface, 1, getBase64PngMapFromAgv, map_header)                                     \
01521     _FUNC(AgvcInterface, 3, previewPngMapFromAgv, map_header, image_width_px, image_height_px)      \
01522     _FUNC(AgvcInterface, 1, sendGridMapToAgv, map)                                                  \
01523     _FUNC(AgvcInterface, 1, sendBase64PngMapToAgv, map)                                             \
01524     _FUNC(AgvcInterface, 1, saveMap, map_header)                                                    \
01525     _FUNC(AgvcInterface, 1, switchMap, map_header)                                                  \
01526     _FUNC(AgvcInterface, 1, deleteMap, map_header)                                                  \
01527     _FUNC(AgvcInterface, 1, deleteMaps, map_headers)                                                \
01528     _FUNC(AgvcInterface, 0, getAllMapVirtualArea)                                                   \
01529     _FUNC(AgvcInterface, 1, getAllMapVirtualAreaOfTargetMap, map_header)                            \
01530     _FUNC(AgvcInterface, 1, addMapVirtualArea, map_virtual_area)                                    \
01531     _FUNC(AgvcInterface, 1, addMapVirtualAreas, map_virtual_areas)                                  \
01532     _FUNC(AgvcInterface, 1, deleteMapVirtualArea, virtual_area_header)                              \
01533     _FUNC(AgvcInterface, 1, deleteMapVirtualAreas, virtual_areas_header)                            \
01534     _FUNC(AgvcInterface, 1, getGridMapAllInfo, map_header)                                          \
01535     _FUNC(AgvcInterface, 1, getPngMapAllInfo, map_header)                                           \
01536     _FUNC(AgvcInterface, 2, setGridMapAllInfo, map_all_info, command_header)                        \
01537     _FUNC(AgvcInterface, 2, setPngMapAllInfo, map_all_info, command_header)                         \
01538     _FUNC(AgvcInterface, 1, deleteMapAllInfo, map_header)                                           \
01539     _FUNC(AgvcInterface, 0, getWifiList)                                                            \
01540     _FUNC(AgvcInterface, 2, connectWifi, ssid, password)                                            \
01541     _FUNC(AgvcInterface, 1, enableHotspot, password)                                                \
01542     _FUNC(AgvcInterface, 0, getIpAddressList)                                                       \
01543     _FUNC(AgvcInterface, 2, changeRunningMode, running_mode, command_header)                        \
01544     _FUNC(AgvcInterface, 1, setControlSpeed, speed)                                                 \
01545     _FUNC(AgvcInterface, 1, setNavGoal, target)                                                     \
01546     _FUNC(AgvcInterface, 0, getNavInfo)                                                             \
01547     _FUNC(AgvcInterface, 0, pauseAgvSpeed)                                                          \
01548     _FUNC(AgvcInterface, 0, resumeAgvSpeed)                                                         \
01549     _FUNC(AgvcInterface, 0, cancelNavigation)                                                       \
01550     _FUNC(AgvcInterface, 2, relocation, init_pose, command_header)                                  \
01551     _FUNC(AgvcInterface, 0, getAgvControllerParametersFile)                                         \
01552     _FUNC(AgvcInterface, 1, setAgvControllerParametersFile, agv_parameters)                         \
01553     _FUNC(AgvcInterface, 0, refreshAgvControllerParametersFile)                                     \
01554     _FUNC(AgvcInterface, 0, resetAgvControllerParametersFile)                                       \
01555     _FUNC(AgvcInterface, 2, checkPowerAndAutoCharge, check_power_header, nav_board_charge_station_id)  \
01556     _FUNC(AgvcInterface, 1, setLeaveBoardTargetStation, leave_board_target_station_header)          \
01557     _FUNC(AgvcInterface, 0, getLeaveBoardTargetStation)                                             \
01558     _FUNC(AgvcInterface, 2, forcedAgvToChargingBoard, forced_charge_header, nav_board_charge_station_id)  \
01559     _FUNC(AgvcInterface, 2, forcedAgvLeaveChargingBoard, forced_leave_header, leave_board_target_station_id) \
01560     _FUNC(AgvcInterface, 1, sendAutoChargingCommand, auto_charging_command)                         \
01561     _FUNC(AgvcInterface, 0, getAgvLogMessage)                                                       \
01562     _FUNC(AgvcInterface, 1, updateSoftware, upgrade_pack_path)                                      \
01563     _FUNC(AgvcInterface, 0, getSoftwareVersionList)                                                 \
01564     _FUNC(AgvcInterface, 1, switchSoftwareVersion, software_version)                                \
01565     _FUNC(AgvcInterface, 1, uninstallSoftware, software_version)                                    \
01566     _FUNC(AgvcInterface, 1, updateFirmware, update_firmware)                                        \
01567     _FUNC(AgvcInterface, 0, getUpdateFirmwareProcess)                                               \
01568     _FUNC(AgvcInterface, 2, startCollectCalibrationData, type, command_header)                      \
01569     _FUNC(AgvcInterface, 2, cancelCollectCalibrationData, type, command_header)                     \
01570     _FUNC(AgvcInterface, 2, startCalibration, type, command_header)                                 \
01571     _FUNC(AgvcInterface, 1, getCalibrationProcessInfo, type)                                        \
01572     _FUNC(AgvcInterface, 0, restartAgv)                                                             \
01573     _FUNC(AgvcInterface, 1, setAgvName, agv_name)                                                   \
01574     _FUNC(AgvcInterface, 2, setPriority, name, ip)                                                  \
01575     _FUNC(AgvcInterface, 0, releasePriority)                                                        \
01576     _FUNC(AgvcInterface, 0, scriptPaused)                                                           \
01577     _FUNC(AgvcInterface, 0, scriptResume)                                                           \
01578     _FUNC(AgvcInterface, 0, scriptStop)                                                             \
01579     _FUNC(AgvcInterface, 0, getScriptStatus)                                                        \
01580     _FUNC(AgvcInterface, 1, setScriptStatus, status)                                                \
01581     _FUNC(AgvcInterface, 1, errorCodeDecoder, error_code)                                           \
01582     _FUNC(AgvcInterface, 0, getCurrentErrorCodes)                                                   \
01583     _FUNC(AgvcInterface, 0, soundLightPrompt)                                                       \
01584     _FUNC(AgvcInterface, 1, isObstacleAhead, detect_distance)                                       \
01585     _FUNC(AgvcInterface, 1, getNearestStation, detect_distance)                                     \
01586     _FUNC(AgvcInterface, 1, autoAlignRailway, command_header)
01587 #endif // AGVC_INTERFACE_H
01588
01589
```

13.5 include/agvc_interface/rpc.h 文件参考

用于RPC 模块的交互，如登录、连接等功能

```
#include <memory>
#include <functional>
#include <agvc_interface/agvc_interface.h>
```

rpc.h 的引用 (Include) 关系图:



类

- class agvc_interface::RpcClient
  RPC 客户端

命名空间

- namespace agvc_interface

宏定义

- #define AGVC_ABI
- #define RpcClient_DECLARES

类型定义

- using agvc_interface::RpcClientPtr = std::shared_ptr<RpcClient>

13.5.1 详细描述

用于RPC 模块的交互，如登录、连接等功能
在文件 rpc.h 中定义.

13.5.2 宏定义说明

AGVC_ABI

```
#define AGVC_ABI
```

在文件 rpc.h 第 7 行定义.

RpcClient_DECLARES

```
#define RpcClient_DECLARES
```

值:

```
_FUNC(RpcClient, 1, setLogHandler, handler)          \
_FUNC(RpcClient, 2, connect, ip, port)               \
_FUNC(RpcClient, 0, disconnect)                      \
_FUNC(RpcClient, 0, hasConnected)                    \
_FUNC(RpcClient, 2, login, usrname, passwd)          \
_FUNC(RpcClient, 0, logout)                          \
_FUNC(RpcClient, 0, hasLogined)                      \
_FUNC(RpcClient, 1, setRequestTimeout, timeout)      \
_FUNC(RpcClient, 1, setExceptionFree, timenableeout) \
_FUNC(RpcClient, 0, errorCode)
```

在文件 rpc.h 第 151 行定义.

13.6 rpc.h

浏览该文件的文档.

```
00001 /** @file  rpc.h
00002  *  @brief 用于 RPC 模块的交互，如登录、连接等功能
00003  */
00004 #ifndef AGVC_INTERFACE_RPC_H
00005 #define AGVC_INTERFACE_RPC_H
00006
00007 #define AGVC_ABI
00008
00009 #include <memory>
00010 #include <functional>
00011
00012 #include <agvc_interface/agvc_interface.h>
00013
00014 namespace agvc_interface {
00015
00016 /// RPC 客户端
00017 class AGVC_ABI RpcClient : public AgvcInterface
00018 {
00019 public:
00020     enum Event
00021     {
00022         Connected,
00023         Disconnected,
00024     };
00025
00026     /**
00027      * @brief RpcClient
00028      * @param mode 0-TCP 1-UDS
00029      */
00030     RpcClient(int mode = 0);
00031     ~RpcClient();
00032
00033     /**
00034      * 设置日志处理器
00035      *
00036      * 此函数可设置自定义的日志处理函数来处理日志消息。 \n
00037      * Agvc SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。
00038      * 如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。
00039      * 这个函数可以将用户自定义的日志系统与 Agvc SDK 默认的日志系统合并。
00040      *
00041      * @note setLogHandler 函数要放在即将触发的日志之前，
00042      * 否则会按照默认的形式输出日志。
00043      *
00044      * @param handler 日志处理函数 \n
00045      * 此日志处理函数的下定义如下: \n
00046      * void handler(int level, const char* filename, int line, const
00047      * std::string& message) \n
00048      * level 表示日志等级 \n
00049      *   0: LOGLEVEL_FATAL 严重的错误 \n
00050      *   1: LOGLEVEL_ERROR      错误 \n
00051      *   2: LOGLEVEL_WARNING    警告 \n
00052      *   3: LOGLEVEL_INFO       通知 \n
00053      *   4: LOGLEVEL_DEBUG      调试 \n
00054      *   5: LOGLEVEL_BACKTRACE  跟踪 \n
00055      * filename 表示文件名 \n
00056      * line 表示代码行号 \n
00057      * message 表示日志信息 \n
00058      * @return 无
00059      */
00060     void setLogHandler(
00061         std::function<void(int /*level*/, const char * /*filename*/, int /*line*/, const std::string & /*message*/)>
00062             handler);
00063
```

```
00064     /**
00065      * 连接到 RPC 服务
00066      *
00067      * @param ip IP 地址
00068      * @param port 端口号，RPC 的端口号是 30104
00069      * @param ip 和 port 为空时，采用 unix domain sockets 通讯方式
00070      * @retval 0 RPC 连接成功
00071      * @retval -8 RPC 连接失败，RPC 连接被拒绝
00072      * @retval -15 RPC 连接失败，SDK 版本与 Server 版本不兼容
00073      */
00074     int connect(const std::string &ip = "", int port = 0);
00075
00076     /**
00077      * 断开 RPC 连接
00078      *
00079      * @retval 0 成功
00080      * @retval -1 失败
00081      */
00082     int disconnect();
00083
00084     /**
00085      * 判断是否连接 RPC
00086      *
00087      * @retval true 已连接 RPC
00088      * @retval false 未连接 RPC
00089      */
00090     bool hasConnected() const;
00091
00092     /**
00093      * 登录
00094      *
00095      * @param usrname 用户名
00096      * @param passwd 密码
00097      * @return 0
00098      */
00099     int login(const std::string &usrname, const std::string &passwd);
00100
00101     /**
00102      * 登出
00103      *
00104      * @return 0
00105      */
00106     int logout();
00107
00108     /**
00109      * 判断是否登录
00110      *
00111      * @retval true 已登录
00112      * @retval false 未登录
00113      */
00114     bool hasLogined();
00115
00116     /**
00117      * 设置 RPC 请求超时时间
00118      *
00119      * @param timeout 请求超时时间，单位 ms
00120      * @return 0
00121      */
00122     int setRequestTimeout(int timeout = 10);
00123
00124     /**
00125      * 设置事件处理
00126      *
00127      * @param cb
00128      * @return
00129      */
00130     int setEventHandler(std::function<void(int /*event*/)> cb);
00131
00132     /**
00133      * 是否关闭异常抛出
00134      *
00135      * @param enable
00136      * @return
00137      */
00138     int setExceptionFree(bool enable);
00139
00140     /**
00141      * 返回错误代码
00142      *
00143      * @return
00144      */
00145     int errorCode() const;
00146 };
00147 using RpcClientPtr = std::shared_ptr<RpcClient>;
00148
00149 } // namespace agvc_interface
00150
```

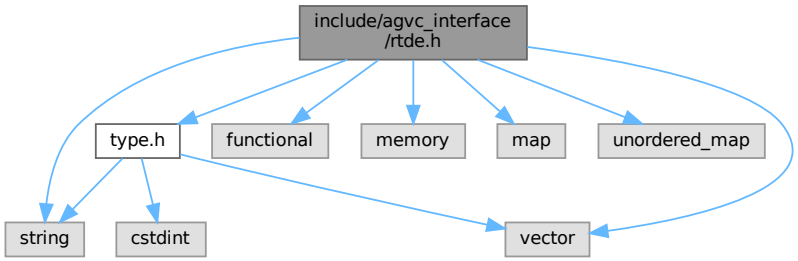
```
00151 #define RpcClient_DECLARES                              \
00152     _FUNC(RpcClient, 1, setLogHandler, handler)         \
00153     _FUNC(RpcClient, 2, connect, ip, port)              \
00154     _FUNC(RpcClient, 0, disconnect)                     \
00155     _FUNC(RpcClient, 0, hasConnected)                   \
00156     _FUNC(RpcClient, 2, login, usrname, passwd)         \
00157     _FUNC(RpcClient, 0, logout)                         \
00158     _FUNC(RpcClient, 0, hasLogined)                     \
00159     _FUNC(RpcClient, 1, setRequestTimeout, timeout)     \
00160     _FUNC(RpcClient, 1, setExceptionFree, timenableeout) \
00161     _FUNC(RpcClient, 0, errorCode)
00162
00163 #endif
```

13.7 include/agvc_interface/rtde.h 文件参考

用于RPC 模块的交互，如订阅、发布等功能

```
#include <string>
#include <vector>
#include <functional>
#include <memory>
#include <map>
#include <unordered_map>
#include "type.h"
```

rtde.h 的引用 (Include) 关系图:



类

- class agvc_interface::OutputBuilder
  向输出数据中增加
- class agvc_interface::InputParser
  解析输入
- class agvc_interface::RtdeClient
  RTDE 客户端

命名空间

- namespace agvc_interface

宏定义

- #define AGVC_ABI
- #define RtdeClient_DECLARES

类型定义

- using agvc_interface::RtdeClientPtr = std::shared_ptr<RtdeClient>

13.7.1 详细描述

用于RPC 模块的交互，如订阅、发布等功能
在文件 rtde.h 中定义.

13.7.2 宏定义说明

AGVC_ABI

```
#define AGVC_ABI
```

在文件 rtde.h 第 7 行定义.

RtdeClient_DECLARES

```
#define RtdeClient_DECLARES
```

值:

```
_FUNC(RtdeClient, 1, setLogHandler, handler)                                \
_FUNC(RtdeClient, 2, connect, ip, port)                                     \
_FUNC(RtdeClient, 0, disconnect)                                            \
_FUNC(RtdeClient, 0, hasConnected)                                          \
_FUNC(RtdeClient, 1, hasConnected1, callback)                               \
_FUNC(RtdeClient, 2, login, usrname, passwd)                                \
_FUNC(RtdeClient, 0, logout)                                                \
_FUNC(RtdeClient, 0, hasLogined)                                            \
_FUNC(RtdeClient, 0, getProtocolVersion)                                    \
_FUNC(RtdeClient, 0, getInputMaps)                                          \
_FUNC(RtdeClient, 0, getOutputMaps)                                         \
_FUNC(RtdeClient, 4, setTopic, to_server, names, freq, expected_chanel)     \
_FUNC(RtdeClient, 2, removeTopic, to_server, chanel)                        \
_FUNC(RtdeClient, 0, getRegisteredInputRecipe)                              \
_FUNC(RtdeClient, 0, getRegisteredOutputRecipe)                             \
_FUNC(RtdeClient, 2, subscribe, chanel, callback)                           \
_FUNC(RtdeClient, 2, publish, chanel, callback)                             \
_FUNC(RtdeClient, 1, setEventHandler, cb)
```

在文件 rtde.h 第 288 行定义.

13.8 rtde.h

浏览该文件的文档.

```
00001 /** @file  rtde.h
00002  *  @brief 用于 RPC 模块的交互，如订阅、发布等功能
00003  */
00004 #ifndef AGVC_SDK_RTDE_H
00005 #define AGVC_SDK_RTDE_H
00006
00007 #define AGVC_ABI
00008
00009 #include <string>
00010 #include <vector>
00011 #include <functional>
00012 #include <memory>
00013 #include <map>
00014 #include <unordered_map>
00015
00016 #include "type.h"
00017
00018 namespace agvc_interface {
00019
00020 class RtdeClient;
00021
00022 /// 向输出数据中增加
00023 class AGVC_ABI OutputBuilder
00024 {
00025 public:
00026     OutputBuilder();
00027     ~OutputBuilder();
00028
00029     OutputBuilder &push(int val);
00030     OutputBuilder &push(double val);
00031     OutputBuilder &push(const std::vector<double> &val);
00032     OutputBuilder &push(const std::tuple<int, bool> &val);
00033     OutputBuilder &push(int16_t &val);
00034     OutputBuilder &push(const std::vector<int16_t> &val);
00035     OutputBuilder &push(const std::vector<int> &val);
00036     OutputBuilder &push(const std::string &val);
00037     OutputBuilder &push(char val);
00038     OutputBuilder &push(const agvc_interface::RtdeRecipe &val);
```

```
00039
00040
00041 private:
00042     friend RtdeClient;
00043     class Impl;
00044     Impl *impl;
00045 };
00046
00047 /// 解析输入
00048 class AGVC_ABI InputParser
00049 {
00050 public:
00051     InputParser();
00052     ~InputParser();
00053
00054     bool popBool();
00055     int popInt32();
00056     int64_t popInt64();
00057     int16_t popInt16();
00058     double popDouble();
00059     char popChar();
00060     std::string popString();
00061     std::vector<int> popVectorInt();
00062     std::vector<int16_t> popVectorInt16();
00063     std::vector<double> popVectorDouble();
00064     std::vector<std::vector<double>> popVectorVectorDouble();
00065     agvc_interface::Pose2d popPose2d();
00066     agvc_interface::RunningInfo popRunningInfo();
00067     agvc_interface::AgvDetails popAgvDetails();
00068     agvc_interface::NavInfo popNavInfo();
00069     std::vector<agvc_interface::Point2d> popVectorPoint2d();
00070     std::vector<agvc_interface::Header> popVectorHeader();
00071
00072 private:
00073     friend RtdeClient;
00074     class Impl;
00075     Impl *impl;
00076 };
00077
00078 /// RTDE 客户端
00079 class AGVC_ABI RtdeClient
00080 {
00081 public:
00082     enum Event
00083     {
00084         Connected,
00085         Disconnected,
00086     };
00087
00088     /**
00089      * RTDE 客户端初始化
00090      *
00091      * @param mode 设置通讯方式， 0 tcp 通讯 1 uds 通讯
00092      */
00093     RtdeClient(int mode = 0);
00094     ~RtdeClient();
00095
00096     /**
00097      * 设置日志处理器
00098      *
00099      * 此函数可设置自定义的日志处理函数来处理日志消息。 \n
00100      * Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。
00101      * 如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。
00102      * 这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。
00103      *
00104      * @note setLogHandler 函数要放在即将触发的日志之前，
00105      * 否则会按照默认的形式输出日志。
00106      *
00107      * @param handler 日志处理函数 \n
00108      * 此日志处理函数的下定义如下: \n
00109      * void handler(int level, const char* filename, int line, const
00110      * std::string& message) \n
00111      * level 表示日志等级 \n
00112      *   0: LOGLEVEL_FATAL 严重的错误 \n
00113      *   1: LOGLEVEL_ERROR      错误 \n
00114      *   2: LOGLEVEL_WARNING    警告 \n
00115      *   3: LOGLEVEL_INFO       通知 \n
00116      *   4: LOGLEVEL_DEBUG      调试 \n
00117      *   5: LOGLEVEL_BACKTRACE  跟踪 \n
00118      * filename 表示文件名 \n
00119      * line 表示代码行号 \n
00120      * message 表示日志信息 \n
00121      * @return 无
00122      */
00123     void setLogHandler(
00124         std::function<void(int /*level*/, const char * /*filename*/, int /*line*/, const std::string & /*message*/)>
00125             handler);
```

```
00126
00127     /**
00128      * 连接到服务器
00129      *
00130      * @param ip IP 地址
00131      * @param port 端口号，RTDE 端口号为 30110
00132      * @retval 0 连接成功
00133      * @retval 1 在执行函数前，已连接
00134      * @retval -1 连接失败
00135      */
00136     int connect(const std::string &ip = "", int port = 0);
00137
00138     /**
00139      * socket 是否已连接
00140      *
00141      * @retval true 已连接 socket
00142      * @retval false 未连接 socket
00143      */
00144     bool hasConnected() const;
00145
00146     /**
00147      * socket 是否已连接
00148      *
00149      * @param callback
00150      * @retval true 已连接 socket
00151      * @retval false 未连接 socket
00152      */
00153     bool hasConnected1(std::function<void(bool)> callback);
00154
00155     /**
00156      * 登录
00157      *
00158      * @param usrname 用户名
00159      * @param passwd 密码
00160      * @retval 0 成功
00161      * @retval -1 失败
00162      */
00163     int login(const std::string &usrname, const std::string &passwd);
00164
00165     /**
00166      * 是否已经登录
00167      *
00168      * @retval true 已登录
00169      * @retval false 未登录
00170      */
00171     bool hasLogined();
00172
00173     /**
00174      * 登出
00175      *
00176      * @return 0
00177      */
00178     int logout();
00179
00180     /**
00181      * 断开连接
00182      *
00183      * @retval 0 成功
00184      * @retval -1 失败
00185      */
00186     int disconnect();
00187
00188     /**
00189      * 获取协议版本号
00190      *
00191      * @return 协议版本号
00192      */
00193     int getProtocolVersion();
00194
00195     /**
00196      * 获取输入列表
00197      *
00198      * @return 输入列表
00199      */
00200     std::map<std::string, int> getInputMaps();
00201
00202     /**
00203      * 获取输出列表
00204      *
00205      * @return 输出列表
00206      */
00207     std::map<std::string, int> getOutputMaps();
00208
00209     /**
00210      * 设置话题
00211      *
00212      * @param to_server 数据流向。
```

```
00213      * true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
00214      * @param names 服务器推送的信息列表
00215      * @param freq 服务器推送信息的频率
00216      * @param expected_chanel 通道。
00217      * 取值范围：0~99，
00218      * 发布不同的话题走不同的通道
00219      * @retval expected_chanel 参数的值 成功
00220      * @retval -1 失败
00221      */
00222     int setTopic(bool to_server, const std::vector<std::string> &names, double freq, int expected_chanel);
00223
00224     /**
00225      * 取消订阅
00226      *
00227      * @param to_server 数据流向
00228      * true 表示客户端给服务器发送消息，false 表示服务器给客户端发送消息
00229      * @param chanel 通道
00230      * @retval 0 成功
00231      * @retval 1 失败
00232      */
00233     int removeTopic(bool to_server, int chanel);
00234
00235     /**
00236      * 获取已注册的输入菜单
00237      *
00238      * @return 已注册的输入菜单
00239      */
00240     std::unordered_map<int, agvc_interface::RtdeRecipe> getRegisteredInputRecipe();
00241
00242     /**
00243      * 获取已注册的输出菜单
00244      *
00245      * @return 已注册的输出菜单
00246      */
00247     std::unordered_map<int, agvc_interface::RtdeRecipe> getRegisteredOutputRecipe();
00248
00249     /**
00250      * 订阅 subscribe from output
00251      *
00252      * @param chanel 通道
00253      * @param callback 回调函数，用于处理订阅的输入信息。\n
00254      * 回调函数的定义如下：
00255      * void callback(InputParser &parser)
00256      * @return 0
00257      */
00258     int subscribe(int chanel, std::function<void(InputParser &)> callback);
00259
00260     /**
00261      * 发布 publish to input
00262      *
00263      * @param chanel 通道
00264      * @param callback 回调函数，用于构建发布的输出信息。\n
00265      * 回调函数的定义如下：
00266      * void callback(OutputBuilder &builder)
00267      * @retval 0 成功
00268      * @retval -1 失败
00269      */
00270     int publish(int chanel, std::function<void(OutputBuilder &)> callback);
00271
00272     /**
00273      * 设置事件处理
00274      *
00275      * @param cb
00276      * @return
00277      */
00278     int setEventHandler(std::function<void(int /*event*/)> cb);
00279
00280 private:
00281     class Impl;
00282     Impl *impl;
00283 };
00284 using RtdeClientPtr = std::shared_ptr<RtdeClient>;
00285
00286 } // namespace agvc_interface
00287
00288 #define RtdeClient_DECLARES                                          \
00289     _FUNC(RtdeClient, 1, setLogHandler, handler)                     \
00290     _FUNC(RtdeClient, 2, connect, ip, port)                          \
00291     _FUNC(RtdeClient, 0, disconnect)                                 \
00292     _FUNC(RtdeClient, 0, hasConnected)                               \
00293     _FUNC(RtdeClient, 1, hasConnected1, callback)                    \
00294     _FUNC(RtdeClient, 2, login, usrname, passwd)                     \
00295     _FUNC(RtdeClient, 0, logout)                                     \
00296     _FUNC(RtdeClient, 0, hasLogined)                                 \
00297     _FUNC(RtdeClient, 0, getProtocolVersion)                         \
00298     _FUNC(RtdeClient, 0, getInputMaps)                               \
00299     _FUNC(RtdeClient, 0, getOutputMaps)                              \
```

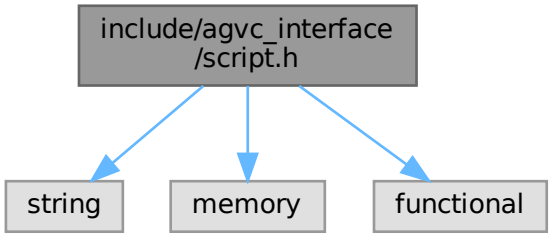
```
00300     _FUNC(RtdeClient, 4, setTopic, to_server, names, freq, expected_chanel) \
00301     _FUNC(RtdeClient, 2, removeTopic, to_server, chanel)                    \
00302     _FUNC(RtdeClient, 0, getRegisteredInputRecipe)                          \
00303     _FUNC(RtdeClient, 0, getRegisteredOutputRecipe)                         \
00304     _FUNC(RtdeClient, 2, subscribe, chanel, callback)                       \
00305     _FUNC(RtdeClient, 2, publish, chanel, callback)                         \
00306     _FUNC(RtdeClient, 1, setEventHandler, cb)
00307
00308 #endif
```

13.9 include/agvc_interface/script.h 文件参考

用于SCRIPT 模块的交互，如向服务器发送脚本

```
#include <string>
#include <memory>
#include <functional>
```

script.h 的引用 (Include) 关系图:



类

- class agvc_interface::ScriptWriter
- class agvc_interface::ScriptClient

  SCRIPT 客户端

命名空间

- namespace agvc_interface

宏定义

- #define AGVC_ABI

类型定义

- using agvc_interface::ScriptClientPtr = std::shared_ptr<ScriptClient>

13.9.1 详细描述

用于SCRIPT 模块的交互，如向服务器发送脚本
在文件 script.h 中定义.

13.9.2 宏定义说明

AGVC_ABI

```
#define AGVC_ABI
```

在文件 script.h 第 6 行定义.

13.10 script.h

浏览该文件的文档.

```
00001 /** @file  script.h
00002  *  @brief 用于 SCRIPT 模块的交互，如向服务器发送脚本
00003  */
00004 #ifndef AUBO_SDK_SCRIPT_H
00005 #define AUBO_SDK_SCRIPT_H
00006 #define AGVC_ABI
00007
00008 #include <string>
00009 #include <memory>
00010 #include <functional>
00011
00012 namespace agvc_interface {
00013
00014 class ScriptWriter
00015 {
00016 public:
00017     virtual ~ScriptWriter() = default;
00018
00019     virtual ScriptWriter &append(const std::string &line)     = 0;
00020     virtual ScriptWriter &append(const char *buf, size_t len) = 0;
00021     virtual ScriptWriter &moveJoint()                         = 0;
00022     virtual ScriptWriter &moveLine()                          = 0;
00023     virtual ScriptWriter &ifCondition()                       = 0;
00024     virtual ScriptWriter &elseCondition()                     = 0;
00025     virtual ScriptWriter &elseIfCondition()                   = 0;
00026     virtual ScriptWriter &whileCondition()                    = 0;
00027     virtual ScriptWriter &end()                               = 0;
00028 };
00029
00030 /// SCRIPT 客户端
00031 class AGVC_ABI ScriptClient
00032 {
00033 public:
00034     enum Event
00035     {
00036         Connected,
00037         Disconnected,
00038     };
00039
00040     ScriptClient(int mode = 0);
00041     ~ScriptClient();
00042
00043     /**
00044      * 设置日志处理器
00045      *
00046      * 此函数可设置自定义的日志处理函数来处理日志消息。 \n
00047      * Aubo SDK 有一套默认的日志系统，按照默认的格式输出到默认的文件。
00048      * 如果用户不希望采用默认的格式或者不希望输出到默认的文件，那就可以通过这个接口重新自定义格式，或者输出路径。
00049      * 这个函数可以将用户自定义的日志系统与 Aubo SDK 默认的日志系统合并。
00050      *
00051      * @note setLogHandler 函数要放在即将触发的日志之前，
00052      * 否则会按照默认的形式输出日志。
00053      *
00054      * @param handler 日志处理函数 \n
00055      * 此日志处理函数的下定义如下: \n
00056      * void handler(int level, const char* filename, int line, const
00057      * std::string& message) \n
00058      * level 表示日志等级 \n
00059      *   0: LOGLEVEL_FATAL 严重的错误 \n
00060      *   1: LOGLEVEL_ERROR      错误 \n
00061      *   2: LOGLEVEL_WARNING    警告 \n
00062      *   3: LOGLEVEL_INFO       通知 \n
00063      *   4: LOGLEVEL_DEBUG      调试 \n
00064      *   5: LOGLEVEL_BACKTRACE  跟踪 \n
00065      * filename 表示文件名 \n
00066      * line 表示代码行号 \n
00067      * message 表示日志信息 \n
00068      * @return 无
00069      */
00070     void setLogHandler(
00071         std::function<void(int /*level*/, const char * /*filename*/, int /*line*/, const std::string & /*message*/)>
00072             handler);
00073
00074     /**
00075      * 连接到服务器
00076      *
00077      * @param ip IP 地址
00078      * @param port 端口号。SCRIPT 端口号为 30004
00079      * @retval 0 连接成功
00080      * @retval 1 在执行函数前，已连接
00081      * @retval -1 连接失败
00082      */
00083     int connect(const std::string &ip = "", int port = 0);
```

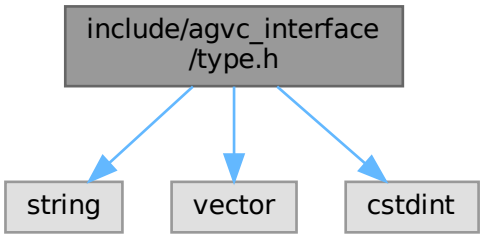
```
00084
00085     /**
00086      * 是否处于连接状态
00087      *
00088      * @retval true 已连接
00089      * @retval false 未连接
00090      */
00091     bool hasConnected() const;
00092
00093     /**
00094      * 登录
00095      *
00096      * @param usrname 用户名
00097      * @param passwd 密码
00098      * @retval 0 成功
00099      * @retval -1 失败
00100      */
00101     int login(const std::string &usrname, const std::string &passwd);
00102
00103     /**
00104      * 返回客户端是否登录
00105      *
00106      * @retval true 已登录
00107      * @retval false 未登录
00108      */
00109     bool hasLogined();
00110
00111     /**
00112      * 登出
00113      *
00114      * @return 0
00115      */
00116     int logout();
00117
00118     /**
00119      * 断开连接
00120      *
00121      * @retval 0 成功
00122      * @retval -1 失败
00123      */
00124     int disconnect();
00125
00126     /**
00127      * 发送脚本文件
00128      *
00129      * 远程调用机器人的脚本
00130      *
00131      * @param path 文件在机器人端的路径
00132      * @retval 0 成功
00133      * @retval -1 失败
00134      */
00135     int sendFile(const std::string &path);
00136
00137     /**
00138      * 发送脚本内容
00139      *
00140      * 调用本地的脚本
00141      *
00142      * @param script 脚本内容
00143      * @retval 0 成功
00144      * @retval -1 失败
00145      */
00146     int sendString(const std::string &script);
00147
00148     /**
00149      * 使用 ScriptWriter 构建服务器脚本
00150      *
00151      * @param chunck_name
00152      * @param cb
00153      * @retval 0 成功
00154      * @retval -1 失败
00155      */
00156     int send(const std::string &chunck_name, std::function<int(ScriptWriter &)> cb);
00157
00158     /**
00159      * 设置服务器脚本的全局变量
00160      *
00161      * @param cb 脚本的全局变量
00162      * @return 0
00163      */
00164     int subscribeVariableUpdate(std::function<void(const std::string &, const std::string &)> cb);
00165
00166     /**
00167      * 设置服务器脚本的错误码
00168      *
00169      * @param cb 脚本的错误码
00170      * @return 0
```

```
00171      */
00172     [[deprecated]] int subscribeScriptError(std::function<void(const std::string &)> cb);
00173
00174     int subscribeScriptError2(std::function<void(const std::string &, const std::string &)> cb);
00175
00176     /**
00177      * 设置事件处理
00178      *
00179      * @param cb
00180      * @return
00181      */
00182     int setEventHandler(std::function<void(int /*event*/)> cb);
00183
00184 private:
00185     class Impl;
00186     Impl *impl;
00187 };
00188 using ScriptClientPtr = std::shared_ptr<ScriptClient>;
00189
00190 } // namespace agvc_interface
00191 #endif
```

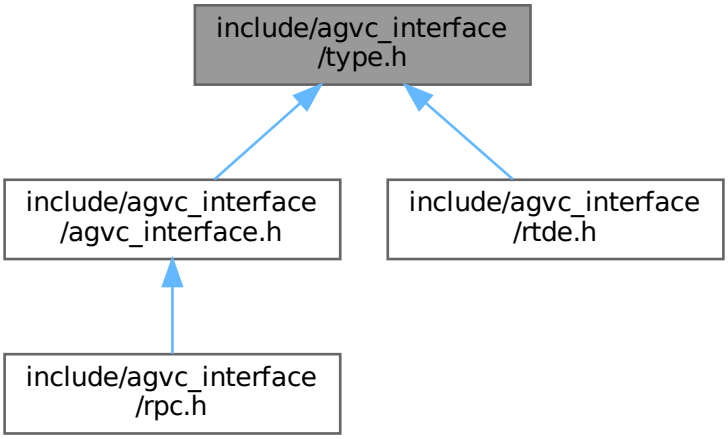
13.11 include/agvc_interface/type.h 文件参考

```
#include <string>
#include <vector>
#include <cstdint>
```

type.h 的引用 (Include) 关系图:



此图展示该文件被哪些文件直接或间接地引用了:



类

- struct agvc_interface::Point2d
  二维点
- struct agvc_interface::Pose2d
  二维位姿
- struct agvc_interface::Speed
  速度
- struct agvc_interface::Header
  头部信息
- struct agvc_interface::AgvDetails
  agv 信息
- struct agvc_interface::RunningInfo
  agv 运行信息
- struct agvc_interface::SaveMapWorkingStatus
  接口 *saveMap* 的工作状态
- struct agvc_interface::SwitchMapWorkingStatus
  接口 *switchMap* 的工作状态
- struct agvc_interface::ChangeModeWorkingStatus
  接口 *changeRunningMode* 的工作状态
- struct agvc_interface::RelocationWorkingStatus
  接口*Relocation* 的工作状态
- struct agvc_interface::SetPngMapAllInfoWorkingStatus
  接口 *setPngMapAllInfo* 的工作状态
- struct agvc_interface::GetPngMapAllInfoWorkingStatus
  接口 *getPngMapAllInfo* 的工作状态
- struct agvc_interface::GetGridMapWorkingStatus
  接口 *getGridMapFromAgv* 的工作状态
- struct agvc_interface::SendGridMapWorkingStatus
  接口 *sendGridMapToAgv* 的工作状态

- struct agvc_interface::GetPngMapWorkingStatus
- struct agvc_interface::SendPngMapWorkingStatus

  接口 *sendBase64PngMapToAgv* 的工作状态
- struct agvc_interface::SetGridMapAllInfoWorkingStatus

  接口 *setGridMapAllInfo* 的工作状态
- struct agvc_interface::GetGridMapAllInfoWorkingStatus

  接口 *getGridMapAllInfo* 的工作状态
- struct agvc_interface::PreviewPngMapWorkingStatus

  接口 *previewPngMapFromAgv* 的工作状态
- struct agvc_interface::AutoAlignRailwayWorkingStatus

  接口 *autoAlignRailway* 的工作状态
- struct agvc_interface::AsyncInterfaceResultStatus

  异步接口运行状态暂时存在 *14* 个异步接口: *saveMap, switchMap, changeRunningMode, relocation, sendBase64PngMapToAgv getGridMapAllInfo, setGridMapAllInfo, sendGridMapToAgv, getGridMap↩FromAgv getPngMapAllInfo, setPngMapAllInfo, getBase64PngMapFromAgv, previewPngMapFromAgv, autoAlignRailway*
- struct agvc_interface::StationMark

  站点
- struct agvc_interface::PathStation

  通过站点表示路径
- struct agvc_interface::MapInfo

  地图信息
- struct agvc_interface::MapVirtualArea

  地图中的虚拟区域
- struct agvc_interface::MapAllInfo

  包含当前地图、当前地图上的站点、当前地图上的路径、当前地图上的虚拟区域信息
- struct agvc_interface::IpAddressInfo

  IP 地址
- struct agvc_interface::AdjustParam

  agv 到点后二次调整位置参数设置
- struct agvc_interface::NavGoalType

  使 *agv* 导航到目标位置（控制信息）
- struct agvc_interface::NavInfo

  导航状态信息 / 自动充电状态信息
- struct agvc_interface::AutoChargingCommand

  自动充电信息
- struct agvc_interface::FirmwareUpdateParam

  配置要更新固件的模块及固件路径
- struct agvc_interface::FirmwareUpdateProcessInfo

  固件更新过程信息
- struct agvc_interface::CalibrationProcessInfo
- struct agvc_interface::RtdeRecipe

  RTDE 菜单
- class agvc_interface::AgvcException

命名空间

- namespace agvc_interface

宏定义

- #define ENUM_AgvcErrorCodes_DECLARES
  接口函数返回值定义
- #define ENUM_ITEM(c, n, ...)
- #define ENUM_ITEM(n, v, s)
- #define ENUM_ITEM(n, v, s)
- #define ENUM_ITEM(n, v, s)

类型定义

- typedef FeedbackStatus agvc_interface::NavStatus
  agv 导航状态
- typedef FeedbackStatus agvc_interface::SaveMapStatus
  保存地图异步接口状态，调用接口 *saveMap* 后的状态
- typedef FeedbackStatus agvc_interface::SwitchMapStatus
  切换地图异步接口状态，调用接口 *switchMap* 后的状态
- typedef FeedbackStatus agvc_interface::ChangeRunningModeStatus
  切换运行模式异步接口状态，调用接口 *changeRunningMode* 后的状态
- typedef FeedbackStatus agvc_interface::RelocationStatus
  重定位状态，调用接口 *relocation* 后的状态
- typedef FeedbackStatus agvc_interface::SetPngMapAllInfoStatus
  上传*Png* 地图全部信息接口状态，调用接口 *setPngMapAllInfo* 后的状态
- typedef FeedbackStatus agvc_interface::GetPngMapAllInfoStatus
  获取*Png* 地图全部信息接口状态，调用接口 *getPngMapAllInfo* 后的状态
- typedef FeedbackStatus agvc_interface::GetGridMapStatus
  获取栅格地图接口状态，调用接口 *getGridMapFromAgv* 后的状态
- typedef FeedbackStatus agvc_interface::SendGridMapStatus
  发送栅格地图接口状态，调用接口 *sendGridMapToAgv* 后的状态
- typedef FeedbackStatus agvc_interface::GetPngMapStatus
  获取 *png* 地图接口状态，调用接口 *getBase64PngMapFromAgv* 后的状态
- typedef FeedbackStatus agvc_interface::SendPngMapStatus
  发送 *png* 地图接口状态，调用接口 *sendBase64PngMapToAgv* 后的状态
- typedef FeedbackStatus agvc_interface::SetGridMapAllInfoStatus
  设置栅格地图全部信息接口状态，调用接口 *setGridMapAllInfo* 后的状态
- typedef FeedbackStatus agvc_interface::GetGridMapAllInfoStatus
  获取栅格地图全部信息接口状态，调用接口 *getGridMapAllInfo* 后的状态
- typedef FeedbackStatus agvc_interface::PreviewPngMapStatus
  获取栅格地图全部信息接口状态，调用接口 *previewPngMapFromAgv* 后的状态
- typedef FeedbackStatus agvc_interface::AutoAlignRailwayStatus
  获取自动上轨接口状态，调用接口 *autoAlignRailway* 后的状态
- typedef FeedbackStatus agvc_interface::CalibrationStatus
  标定状态
- typedef MapInfo agvc_interface::OccupancyGridMap
  栅格地图
- typedef MapInfo agvc_interface::Base64PngMap
  Base64 编码的 *png* 地图图片

枚举

- enum agvc_interface::AgvcErrorCodes : int { agvc_interface::ENUM_AgvcErrorCodes_DECLARES }
- enum class agvc_interface::RunningMode { agvc_interface::NONE = 0 , agvc_interface::START = 1 , agvc_interface::MAPPING = 2 , agvc_interface::NAVIGA = 3 , agvc_interface::CHARGING = 4 , agvc_interface::MAINTAIN = 5 , agvc_interface::STOP = 6 }
  agv 运行模式
- enum class agvc_interface::PathShape { agvc_interface::NONE = 0 , agvc_interface::LINE = 1 , agvc_interface::ARC = 2 , agvc_interface::BEZIER = 3 }
  路径的形状
- enum class agvc_interface::MapFormat { agvc_interface::NONE = 0 , agvc_interface::OCCUPANCY_GRID = 1 , agvc_interface::BASE64_PNG = 2 }
  地图格式
- enum class agvc_interface::MapVirtualAreaType { agvc_interface::NONE = 0 , agvc_interface::OBSTACLE = 1 , agvc_interface::SLOW_DOWN = 2 , agvc_interface::INFLATE = 3 }
  地图中虚拟区域的类型
- enum class agvc_interface::MapVirtualAreaShape { agvc_interface::NONE = 0 , agvc_interface::LINE = 1 , agvc_interface::ARC = 2 , agvc_interface::CIRCLE = 3 , agvc_interface::POLYGON = 4 }
  地图中虚拟区域的形状
- enum class agvc_interface::NavType { agvc_interface::NONE = 0 , agvc_interface::FREE_TO_POSE = 1 , agvc_interface::FREE_TO_STATION = 2 , agvc_interface::PATH_TO_STATION = 3 }
  agv 导航类型
- enum class agvc_interface::AutoChargingType { agvc_interface::NONE = 0 , agvc_interface::LOW_POWER_TO_BOARD = 1 , agvc_interface::HIGH_POWER_ = 2 , agvc_interface::FORCE_TO_BOARD = 3 , agvc_interface::FORCE_LEAVE_BOARD = 4 }
  自动充电类型
- enum class agvc_interface::FeedbackStatus { agvc_interface::NONE = 0 , agvc_interface::FINISHED = 1 , agvc_interface::FAILED = 2 , agvc_interface::RUNNING = 3 , agvc_interface::PAUSED = 4 , agvc_interface::CANCELED = 5 }
  一个操作执行后反馈的状态
- enum class agvc_interface::FirmwareUpdateMode { agvc_interface::NONE = 0 , agvc_interface::UPDATE = 1 , agvc_interface::FORCED_UPDATE = 2 , agvc_interface::NOT_UPDATE = 3 }
  模块固件更新配置
- enum class agvc_interface::ScriptRuntimeState { agvc_interface::RUNNING = 0 , agvc_interface::PAUSED = 1 , agvc_interface::STOPPED = 2 , agvc_interface::ABORTING = 3 }
- enum class agvc_interface::CalibrationType { agvc_interface::NONE = 0 , agvc_interface::LASER_ODOM = 1 }
  标定类型
- enum agvc_interface::error_type { agvc_interface::parse_error = -32700 , agvc_interface::invalid_request = -32600 , agvc_interface::method_not_fou = -32601 , agvc_interface::invalid_params = -32602 , agvc_interface::internal_error = -32603 , agvc_interface::server_error , agvc_interface::invalid }
- enum agvc_interface::ExceptionCode { agvc_interface::EC_DISCONNECTED = -1 , agvc_interface::EC_NOT_LOGINED = -2 , agvc_interface::EC_IN = -3 , agvc_interface::EC_REQUEST_BUSY = -4 , agvc_interface::EC_SEND_FAILED = -5 , agvc_interface::EC_RECV_TIMEOUT = -6 , agvc_interface::EC_RE = -7 , agvc_interface::EC_PARSE_ERROR = -8 , agvc_interface::EC_INVALID_REQUEST = -9 , agvc_interface::EC_METHOD_NOT_FOUND = -10 , agvc_interface::EC_INVALID_PARAMS = -11 , agvc_interface::EC_INTERNAL_ERROR = -12 , agvc_interface::EC_SERVER_ERROR = -13 , agvc_interface::EC_INVALID = -14 }

函数

- const char * agvc_interface::returnValue2Str (int retval)

13.11.1 宏定义说明

ENUM_AgvcErrorCodes_DECLARES

```
#define ENUM_AgvcErrorCodes_DECLARES
```

接口函数返回值定义
+ 数为警告，负数为错误，0 为没有错误也没有警告
在文件 type.h 第 12 行定义.

ENUM_ITEM [1/4]

```
#define ENUM_ITEM(
            c,
            n,
            ...)
```

值:

```
c = n,
```

在文件 type.h 第 70 行定义.

ENUM_ITEM [2/4]

```
#define ENUM_ITEM(
            n,
            v,
            s)
```

值:

```
if (retval == v)       \
    index = n##_INDEX;
```

在文件 type.h 第 70 行定义.

ENUM_ITEM [3/4]

```
#define ENUM_ITEM(
            n,
            v,
            s)
```

值:

```
n##_INDEX,
```

在文件 type.h 第 70 行定义.

ENUM_ITEM [4/4]

```
#define ENUM_ITEM(
            n,
            v,
            s)
```

值:

```
s,
```

在文件 type.h 第 70 行定义.

13.12 type.h

浏览该文件的文档.

```
00001 #ifndef AGVC_INTERFACE_TYPE_H
00002 #define AGVC_INTERFACE_TYPE_H
00003
00004 #include <string>
```

```
00005 #include <vector>
00006 #include <cstdint>
00007
00008 namespace agvc_interface {
00009 /// 接口函数返回值定义
00010 ///
00011 /// + 数为警告，负数为错误，0 为没有错误也没有警告
00012 #define ENUM_AgvcErrorCodes_DECLARES                                    \
00013     ENUM_ITEM(AGVC_OK, 0, " 成功")                                       \
00014     ENUM_ITEM(AGVC_BAD_STATE, 1, " 状态错误")                            \
00015     ENUM_ITEM(AGVC_QUEUE_FULL, 2, " 规划队列满")                         \
00016     ENUM_ITEM(AGVC_BUSY, 3, " 上一条指令正在执行中")                     \
00017     ENUM_ITEM(AGVC_TIMEOUT, 4, " 超时")                                 \
00018     ENUM_ITEM(AGVC_INVL_ARGUMENT, 5, " 参数无效")                        \
00019     ENUM_ITEM(AGVC_NOT_IMPLETEMENT, 6, " 接口未实现")                    \
00020     ENUM_ITEM(AGVC_NO_ACCESS, 7, " 无法访问")                           \
00021     ENUM_ITEM(AGVC_CONN_REFUSED, 8, " 连接被拒绝")                       \
00022     ENUM_ITEM(AGVC_CONN_RESET, 9, " 连接被重置")                         \
00023     ENUM_ITEM(AGVC_INPROGRESS, 10, " 正在执行中")                        \
00024     ENUM_ITEM(AGVC_EIO, 11, "input/output error")                       \
00025     ENUM_ITEM(AGVC_NOBUFFS, 12, "")                                     \
00026     ENUM_ITEM(AGVC_REQUEST_IGNORE, 13, " 本次请求被忽略")                \
00027     ENUM_ITEM(AGVC_ALGORITHM_PLAN_FAILED, 14, " 运动规划算法错误")       \
00028     ENUM_ITEM(AGVC_VERSION_INCOMPAT, 15, " 接口版本不匹配")              \
00029     ENUM_ITEM(AGVC_DIMENSION_ERR, 16, " 输入参数维数不对")               \
00030     ENUM_ITEM(AGVC_SINGULAR_ERR, 17, " 输入的配置可能会奇异")            \
00031     ENUM_ITEM(AGVC_POS_BOUND_ERR, 18, " 输入的位置边界超过极限范围")     \
00032     ENUM_ITEM(AGVC_INIT_POS_ERR, 19, " 初始位置输入不合理")              \
00033     ENUM_ITEM(AGVC_ELP_SETTING_ERR, 20, " 包络体设置错误")               \
00034     ENUM_ITEM(AGVC_TRAJ_GEN_FAIL, 21, " 轨迹生成失败")                   \
00035     ENUM_ITEM(AGVC_TRAJ_SELF_COLLISION, 22, " 轨迹自碰撞")               \
00036     ENUM_ITEM(AGVC_IK_NO_CONVERGE, 23, " 逆解计算不收敛，计算出错")      \
00037     ENUM_ITEM(AGVC_IK_OUT_OF_RANGE, 24, " 逆解计算超出机器人最大限制")   \
00038     ENUM_ITEM(AGVC_IK_CONFIG_DISMATCH, 25, " 逆解输入配置存在错误")      \
00039     ENUM_ITEM(AGVC_IK_JACOBIAN_FAILED, 26, " 逆解雅可比矩阵计算失败")    \
00040     ENUM_ITEM(AGVC_IK_NO_SOLU, 27, " 目标点存在解析解，但均不满足选解条件") \
00041     ENUM_ITEM(AGVC_IK_UNKOWN_ERROR, 28, " 逆解返回未知类型错误")         \
00042     ENUM_ITEM(AGVC_ERR_UNKOWN, 99999, "Unkown error occurred.")         \
00043     ENUM_ITEM(STATION_SRV_OK, 100, " 站点操作成功")                      \
00044     ENUM_ITEM(STATION_SRV_NOT_FOUND, -101, " 站点服务器未响应")          \
00045     ENUM_ITEM(STATION_CREAT_FAILED, -102, " 站点创建失败")               \
00046     ENUM_ITEM(STATION_DELETE_FAILED, -103, " 站点删除失败")              \
00047     ENUM_ITEM(PATH_SRV_OK, 200, " 路径操作成功")                         \
00048     ENUM_ITEM(PATH_SRV_NOT_FOUND, -201, " 路径服务器未响应")             \
00049     ENUM_ITEM(PATH_CREAT_FAILED, -202, " 路径创建失败")                  \
00050     ENUM_ITEM(PATH_DELETE_FAILED, -203, " 路径删除失败")                 \
00051     ENUM_ITEM(MAP_SRV_OK, 300, " 地图操作成功")                          \
00052     ENUM_ITEM(MAP_SRV_NOT_FOUND, -301, " 地图服务器未响应")              \
00053     ENUM_ITEM(MAP_HEADER_LIST_FAILED, -302, " 获取地图 header list 失败") \
00054     ENUM_ITEM(MAP_HEADER_FAILED, -303, " 获取地图 header 失败")          \
00055     ENUM_ITEM(MAP_SET_FAILED, -304, " 地图设置失败")                     \
00056     ENUM_ITEM(MAP_SAVE_FAILED, -305, " 地图保存失败")                    \
00057     ENUM_ITEM(MAP_SWITCH_FAILED, -306, " 地图切换失败")                  \
00058     ENUM_ITEM(MAP_DELETE_FAILED, -307, " 地图删除失败")                  \
00059     ENUM_ITEM(MAP_VIRTUAL_SRV_OK, 301, " 虚拟区操作成功")                \
00060     ENUM_ITEM(MAP_VIRTUAL_SRV_NOT_FOUND, -308, " 虚拟区服务器未响应")    \
00061     ENUM_ITEM(MAP_VIRTUAL_SET_PARAM_WRONG, -309, " 虚拟区设置参数错误")  \
00062     ENUM_ITEM(MAP_VIRTUAL_ADD_FAILED, -310, " 添加虚拟区失败")           \
00063     ENUM_ITEM(MAP_VIRTUAL_DELETE_FAILED, -311, " 删除虚拟区失败")        \
00064     ENUM_ITEM(IP_SRV_OK, 400, "IP 操作成功")                             \
00065     ENUM_ITEM(IP_SRV_NOT_FOUND, -401, "ip 设置服务器未响应")             \
00066     ENUM_ITEM(IP_NETWORK_NULL, -402, "ip 地址网卡名称为空")              \
00067     ENUM_ITEM(IP_ADDRESS_NULL, -403, "ip 地址为空")                      \
00068     ENUM_ITEM(IP_SET_FAILED, -404, "ip 设置失败")
00069
00070 #define ENUM_ITEM(c, n, ...) c = n,
00071 enum AgvcErrorCodes : int
00072 {
00073     ENUM_AgvcErrorCodes_DECLARES
00074 };
00075
00076 /**
00077  * agv 运行模式
00078  */
00079 enum class RunningMode
00080 {
00081     NONE     = 0,
00082     START    = 1, ///< 开始模式
00083     MAPPING  = 2, ///< 建图模式
00084     NAVIGATE = 3, ///< 导航模式
00085     CHARGING = 4, ///< 自动充电模式
00086     MAINTAIN = 5, ///< 维护模式
00087     STOP     = 6, ///< 停止模式
00088 };
00089
00090 /**
00091  * 路径的形状
```

```
00092  */
00093 enum class PathShape
00094 {
00095     NONE   = 0,
00096     LINE   = 1, ///< 直线
00097     ARC    = 2, ///< 圆弧
00098     BEZIER = 3, ///< 贝塞尔曲线
00099                 ///< B_SPLINE = 4  // B 样条曲线
00100 };
00101
00102 /**
00103  * 地图格式
00104  */
00105 enum class MapFormat
00106 {
00107     NONE           = 0,
00108     OCCUPANCY_GRID = 1, ///< 占用栅格地图
00109     BASE64_PNG     = 2, ///< base64 编码的 png 地图
00110 };
00111
00112 /**
00113  * 地图中虚拟区域的类型
00114  */
00115 enum class MapVirtualAreaType
00116 {
00117     NONE      = 0,
00118     OBSTACLE  = 1, ///< 虚拟墙
00119     SLOW_DOWN = 2, ///< 减速区
00120     INFLATE   = 3, ///< 膨胀区
00121 };
00122
00123 /**
00124  * 地图中虚拟区域的形状
00125  */
00126 enum class MapVirtualAreaShape
00127 {
00128     NONE    = 0,
00129     LINE    = 1, ///< 直线
00130     ARC     = 2, ///< 圆弧
00131     CIRCLE  = 3, ///< 圆形
00132     POLYGON = 4, ///< 多边形
00133 };
00134
00135 /**
00136  * agv 导航类型
00137  */
00138 enum class NavType
00139 {
00140     NONE            = 0, ///< 空
00141     FREE_TO_POSE    = 1, ///< 自由导航到任意点
00142     FREE_TO_STATION = 2, ///< 自由导航到站点
00143     PATH_TO_STATION = 3, ///< 路径导航到站点
00144 };
00145
00146 /**
00147  * @brief 自动充电类型
00148  * @since 0.7.0
00149  * @note 高电量阈值与低电量阈值在对外配置参数中进行修改
00150  */
00151 enum class AutoChargingType
00152 {
00153     NONE                   = 0, ///< 空
00154     LOW_POWER_TO_BOARD     = 1, ///< 低电量自动上桩；如果当前电量低于低电量阈值，开始上桩；否则，不执行上桩；
00155     HIGH_POWER_LEAVE_BOARD = 2, ///< 高电量自动下桩；如果当前电量高于高电量阈值，开始下桩；否则，不执行下桩；
00156     FORCE_TO_BOARD         = 3, ///< 强制自动上桩；无视当前电量
00157     FORCE_LEAVE_BOARD      = 4, ///< 强制自动下桩；无视当前电量
00158 };
00159
00160 /**
00161  * 一个操作执行后反馈的状态
00162  */
00163 enum class FeedbackStatus
00164 {
00165     NONE     = 0, ///< 空
00166     FINISHED = 1, ///< 成功
00167     FAILED   = 2, ///< 失败
00168     RUNNING  = 3, ///< 正在运行
00169     PAUSED   = 4, ///< 暂停
00170     CANCELED = 5, ///< 取消
00171 };
00172
00173 /**
00174  * 模块固件更新配置
00175  */
00176 enum class FirmwareUpdateMode
00177 {
00178     NONE          = 0, ///< 空 默认不更新固件
```

```
00179     UPDATE        = 1, ///< 更新固件
00180     FORCED_UPDATE = 2, ///< 强制更新固件
00181     NOT_UPDATE    = 3, ///< 不更新固件
00182 };
00183
00184 // 脚本状态
00185 enum class ScriptRuntimeState
00186 {
00187     RUNNING  = 0, // 正在运行中
00188     PAUSED   = 1, // 暂停
00189     STOPPED  = 2, // 已停止
00190     ABORTING = 3, // 中止
00191 };
00192
00193 /**
00194  * 标定类型
00195  */
00196 enum class CalibrationType
00197 {
00198     NONE       = 0, ///< 空
00199     LASER_ODOM = 1, ///< 激光-码盘里程计标定
00200 };
00201
00202 /**
00203  * @brief agv 导航状态
00204  * @details
00205  * NONE     = 0, // 空 \n
00206  * FINISHED = 1, // 向站点导航完成 \n
00207  * FAILED   = 2, // 向站点导航失败 \n
00208  * RUNNING  = 3, // 正在向站点导航 \n
00209  * PAUSED   = 4, // 向站点导航暂停 \n
00210  * CANCELED = 5, // 向站点导航取消
00211  */
00212 typedef FeedbackStatus NavStatus;
00213
00214 /**
00215  * @brief 保存地图异步接口状态，调用接口 saveMap 后的状态
00216  * @details
00217  * NONE     = 0, // 保存地图接口未执行 \n
00218  * FINISHED = 1, // 保存地图接口执行完毕 \n
00219  * FAILED   = 2, // 无效值 \n
00220  * RUNNING  = 3, // 保存地图中 \n
00221  * PAUSED   = 4, // 无效值 \n
00222  * CANCELED = 5, // 无效值
00223  */
00224 typedef FeedbackStatus SaveMapStatus;
00225
00226 /**
00227  * @brief 切换地图异步接口状态，调用接口 switchMap 后的状态
00228  * @details
00229  * NONE     = 0, // 切换地图接口未执行 \n
00230  * FINISHED = 1, // 切换地图接口执行完毕 \n
00231  * FAILED   = 2, // 无效值 \n
00232  * RUNNING  = 3, // 切换地图中 \n
00233  * PAUSED   = 4, // 无效值 \n
00234  * CANCELED = 5, // 无效值
00235  */
00236 typedef FeedbackStatus SwitchMapStatus;
00237
00238 /**
00239  * @brief 切换运行模式异步接口状态，调用接口 changeRunningMode 后的状态
00240  * @details
00241  * NONE     = 0, // 切换运行模式接口未执行 \n
00242  * FINISHED = 1, // 切换运行模式接口执行完毕 \n
00243  * FAILED   = 2, // 无效值 \n
00244  * RUNNING  = 3, // 切换运行模式中 \n
00245  * PAUSED   = 4, // 无效值 \n
00246  * CANCELED = 5, // 无效值
00247  */
00248 typedef FeedbackStatus ChangeRunningModeStatus;
00249
00250 /**
00251  * @brief 重定位状态，调用接口 relocation 后的状态
00252  * @details
00253  * NONE     = 0, // 重定位接口未执行 \n
00254  * FINISHED = 1, // 重定位接口执行完毕 \n
00255  * FAILED   = 2, // 无效值 \n
00256  * RUNNING  = 3, // 重定位中 \n
00257  * PAUSED   = 4, // 无效值 \n
00258  * CANCELED = 5, // 无效值
00259  */
00260 typedef FeedbackStatus RelocationStatus;
00261
00262 /**
00263  * @brief 上传 Png 地图全部信息接口状态，调用接口 setPngMapAllInfo 后的状态
00264  * @details
00265  * NONE     = 0, // 设置地图信息接口未执行 \n
```

```
00266  * FINISHED = 1, // 设置地图信息接口执行完毕 \n
00267  * FAILED   = 2, // 无效值 \n
00268  * RUNNING  = 3, // 地图信息上传中 \n
00269  * PAUSED   = 4, // 无效值 \n
00270  * CANCELED = 5, // 无效值
00271  */
00272 typedef FeedbackStatus SetPngMapAllInfoStatus;
00273
00274 /**
00275  * @brief 获取 Png 地图全部信息接口状态，调用接口 getPngMapAllInfo 后的状态
00276  * @details
00277  * NONE     = 0, // 获取地图信息接口未执行 \n
00278  * FINISHED = 1, // 获取地图信息接口执行完毕 \n
00279  * FAILED   = 2, // 无效值 \n
00280  * RUNNING  = 3, // 地图信息获取中 \n
00281  * PAUSED   = 4, // 无效值 \n
00282  * CANCELED = 5, // 无效值
00283  */
00284 typedef FeedbackStatus GetPngMapAllInfoStatus;
00285
00286 /**
00287  * @brief 获取栅格地图接口状态，调用接口 getGridMapFromAgv 后的状态
00288  * @details
00289  * NONE     = 0, // 获取栅格地图接口未执行 \n
00290  * FINISHED = 1, // 获取栅格地图接口执行完毕 \n
00291  * FAILED   = 2, // 无效值 \n
00292  * RUNNING  = 3, // 栅格地图获取中 \n
00293  * PAUSED   = 4, // 无效值 \n
00294  * CANCELED = 5, // 无效值
00295  */
00296 typedef FeedbackStatus GetGridMapStatus;
00297
00298 /**
00299  * @brief 发送栅格地图接口状态，调用接口 sendGridMapToAgv 后的状态
00300  * @details
00301  * NONE     = 0, // 发送栅格地图接口未执行 \n
00302  * FINISHED = 1, // 发送栅格地图接口执行完毕 \n
00303  * FAILED   = 2, // 无效值 \n
00304  * RUNNING  = 3, // 发送栅格地图中 \n
00305  * PAUSED   = 4, // 无效值 \n
00306  * CANCELED = 5, // 无效值
00307  */
00308 typedef FeedbackStatus SendGridMapStatus;
00309
00310 /**
00311  * @brief 获取 png 地图接口状态，调用接口 getBase64PngMapFromAgv 后的状态
00312  * @details
00313  * NONE     = 0, // 获取 png 地图接口未执行 \n
00314  * FINISHED = 1, // 获取 png 地图接口执行完毕 \n
00315  * FAILED   = 2, // 无效值 \n
00316  * RUNNING  = 3, // 获取 png 地图中 \n
00317  * PAUSED   = 4, // 无效值 \n
00318  * CANCELED = 5, // 无效值
00319  */
00320 typedef FeedbackStatus GetPngMapStatus;
00321
00322 /**
00323  * @brief 发送 png 地图接口状态，调用接口 sendBase64PngMapToAgv 后的状态
00324  * @details
00325  * NONE     = 0, // 发送 png 地图接口未执行 \n
00326  * FINISHED = 1, // 发送 png 地图接口执行完毕 \n
00327  * FAILED   = 2, // 无效值 \n
00328  * RUNNING  = 3, // png 地图发送中 \n
00329  * PAUSED   = 4, // 无效值 \n
00330  * CANCELED = 5, // 无效值
00331  */
00332 typedef FeedbackStatus SendPngMapStatus;
00333
00334 /**
00335  * @brief 设置栅格地图全部信息接口状态，调用接口 setGridMapAllInfo 后的状态
00336  * @details
00337  * NONE     = 0, // 发送栅格地图全部信息接口未执行 \n
00338  * FINISHED = 1, // 发送栅格地图全部信息接口执行完毕 \n
00339  * FAILED   = 2, // 无效值 \n
00340  * RUNNING  = 3, // 栅格地图发送中 \n
00341  * PAUSED   = 4, // 无效值 \n
00342  * CANCELED = 5, // 无效值
00343  */
00344 typedef FeedbackStatus SetGridMapAllInfoStatus;
00345
00346 /**
00347  * @brief 获取栅格地图全部信息接口状态，调用接口 getGridMapAllInfo 后的状态
00348  * @details
00349  * NONE     = 0, // 获取栅格地图全部信息接口未执行 \n
00350  * FINISHED = 1, // 获取栅格地图全部信息接口执行完毕 \n
00351  * FAILED   = 2, // 无效值 \n
00352  * RUNNING  = 3, // 栅格地图全部信息获取中 \n
```

```
00353  * PAUSED   = 4, // 无效值 \n
00354  * CANCELED = 5, // 无效值
00355  */
00356 typedef FeedbackStatus GetGridMapAllInfoStatus;
00357
00358 /**
00359  * @brief 获取栅格地图全部信息接口状态，调用接口 previewPngMapFromAgv 后的状态
00360  * @details
00361  * NONE     = 0, // 预览地图接口未执行 \n
00362  * FINISHED = 1, // 预览地图接口执行完毕 \n
00363  * FAILED   = 2, // 无效值 \n
00364  * RUNNING  = 3, // 预览地图接口获取中 \n
00365  * PAUSED   = 4, // 无效值 \n
00366  * CANCELED = 5, // 无效值
00367  */
00368 typedef FeedbackStatus PreviewPngMapStatus;
00369
00370 /**
00371  * @brief 获取自动上轨接口状态，调用接口 autoAlignRailway 后的状态
00372  * @details
00373  * NONE     = 0, // 自动上轨接口未执行 \n
00374  * FINISHED = 1, // 自动上轨接口执行完毕 \n
00375  * FAILED   = 2, // 无效值 \n
00376  * RUNNING  = 3, // 自动上轨接口执行中 \n
00377  * PAUSED   = 4, // 无效值 \n
00378  * CANCELED = 5, // 无效值
00379  */
00380 typedef FeedbackStatus AutoAlignRailwayStatus;
00381
00382 /**
00383  * @brief 标定状态
00384  * @details
00385  * NONE     = 0, // 空 \n
00386  * FINISHED = 1, // 标定完成 \n
00387  * FAILED   = 2, // 标定失败 \n
00388  * RUNNING  = 3, // 正在采集数据 \n
00389  * PAUSED   = 4, // 无效值 \n
00390  * CANCELED = 5, // 取消标定
00391  */
00392 typedef FeedbackStatus CalibrationStatus;
00393
00394 /**
00395  * 二维点
00396  */
00397 struct Point2d
00398 {
00399     float x; ///< 单位 m
00400     float y; ///< 单位 m
00401 };
00402
00403 /**
00404  * 二维位姿
00405  */
00406 struct Pose2d
00407 {
00408     double x;   ///< 单位 m
00409     double y;   ///< 单位 m
00410     double yaw; ///< 单位 rad
00411 };
00412
00413 /**
00414  * 速度
00415  */
00416 struct Speed
00417 {
00418     double v; ///< 单位 m/s
00419     double w; ///< 单位 rad/s
00420 };
00421
00422 /**
00423  * 头部信息
00424  */
00425 struct Header
00426 {
00427     std::string id;      ///< uuid
00428     std::string name;    ///< 名称
00429     int64_t stamp;       ///< 时间戳，UTC 时间（1970 年 01 月 01 日 00 时 00 分 00 秒 ~ 至今），单位：纳秒
00430     std::string map_id;  ///< 当前地图的 id，形式为 uuid：例如 a5e7c6de-4463-443e-8b25-ccbfd6a27aee，
00431                          ///< 与地图文件名字相同
00432     int error_code = -1; ///< 错误码
00433 };
00434
00435 /**
00436  * agv 信息
00437  */
00438 struct AgvDetails
00439 {
```

```
00440     std::string name;                 ///< agv 的名称
00441     std::string version;              ///< 当前控制程序的版本号
00442     std::string MFD;                  ///< 生产日期
00443     std::string SN;                   ///< S/N 码
00444     int communication_version;        ///< bridge 版本信息
00445     int battery_version;              ///< 电池版本信息
00446     int loop_times;                   ///< 电池循环次数
00447     float surplus_capacity;           ///< 电池剩余容量
00448     int left_motor_firmware_version;  ///< 左电机固件版本
00449     int right_motor_firmware_version; ///< 右电机固件版本
00450     int master_board_firmware_version; ///< 接口板固件版本信息
00451     int hardware_abstraction_version; ///< 硬件抽象层版本信息
00452     int error_code_version;           ///< 错误码版本
00453     float max_speed;                  ///< agv 允许运行的最大线速度
00454     float max_angular;                ///< agv 允许运行的最大角速度
00455 };
00456
00457 /**
00458  * agv 运行信息
00459  */
00460 struct RunningInfo
00461 {
00462     Header header;               ///< agv 的 SN 码，agv 的名称，当前时间，地图的 id
00463     RunningMode running_mode;    ///< agv 当前运行模式
00464     Speed current_speed;         ///< 当前行驶速度 {v w}，单位 {m/s rad/s}
00465     int8_t location_score;       ///< 当前定位评分，[0-100]
00466     double total_dist;           ///< 累计行驶里程，单位 m
00467     double current_dist;         ///< 本次累计行驶里程，单位 m
00468     double total_running_time;   ///< 累计运行时间，单位 s
00469     double current_running_time; ///< 本次运行时间，单位 s
00470     double battery_voltage;      ///< 电池电压，单位 V
00471     double battery_current;      ///< 电池电流，单位 A
00472     int8_t remaining_voltage;    ///< 剩余电量，百分比 % [0-100]
00473     int8_t battery_status;       ///< 电池状态，0-无,1-放电，2-充电，3-充满，4-急需充电
00474     int8_t safety_edge_status;   ///< 安全触边状态，0-无,1-左前侧碰撞，2-右前碰撞，3-左后碰撞，4-右后碰撞
00475     bool low_battery;            ///< 是否电量低
00476     bool pause;                  ///< 是否处于暂停状态
00477     bool emergency_stop;         ///< 是否处于急停状态
00478     bool blocked;                ///< 是否被障碍物遮挡
00479     int8_t obstacle_level;       ///< 停障等级，0-未停障，1-近距离停障，2-中等距离停障，3-远距离停障
00480     bool localization_loss;      ///< 是否定位丢失
00481     bool fault_alarm;            ///< 故障报警（电机故障、传感器故障、电池故障），具体故障信息可在日志中查看
00482     bool break_release_flag;     ///< 是否释放刹车，true 代表释放，false 代表未释放
00483 };
00484
00485 /**
00486  * 接口 saveMap 的工作状态
00487  */
00488 struct SaveMapWorkingStatus
00489 {
00490     Header header;        ///< header.id 代表下发的 saveMap 指令 id
00491     SaveMapStatus status; ///< 接口 SaveMap 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00492 };
00493
00494 /**
00495  * 接口 switchMap 的工作状态
00496  */
00497 struct SwitchMapWorkingStatus
00498 {
00499     Header header;          ///< header.id 代表下发的 switchMap 指令 id
00500     SwitchMapStatus status; ///< 接口 switchMap 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00501 };
00502
00503 /**
00504  * 接口 changeRunningMode 的工作状态
00505  */
00506 struct ChangeModeWorkingStatus
00507 {
00508     Header header;                  ///< header.id 代表下发的 changeRunningMode 指令 id
00509     ChangeRunningModeStatus status; ///< 接口 changeRunningMode 的运行状态
00510                                     ///< NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00511 };
00512
00513 /**
00514  * 接口 Relocation 的工作状态
00515  */
00516 struct RelocationWorkingStatus
00517 {
00518     Header header;          ///< header.id 代表下发的 Relocation 指令 id
00519     RelocationStatus status; ///< 接口 Relocation 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00520 };
00521
00522 /**
00523  * 接口 setPngMapAllInfo 的工作状态
00524  */
00525 struct SetPngMapAllInfoWorkingStatus
00526 {
```

```
00527     Header header;                ///< header.id 代表下发的 setPngMapAllInfo 指令 id
00528     SetPngMapAllInfoStatus status; ///< 接口 setPngMapAllInfo 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行
      结束
00529 };
00530
00531 /**
00532  * 接口 getPngMapAllInfo 的工作状态
00533  */
00534 struct GetPngMapAllInfoWorkingStatus
00535 {
00536     Header header;                ///< header.id 代表下发的 getPngMapAllInfo 指令 id
00537     GetPngMapAllInfoStatus status; ///< 接口 getPngMapAllInfo 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行
      结束
00538 };
00539
00540 /**
00541  * 接口 getGridMapFromAgv 的工作状态
00542  */
00543 struct GetGridMapWorkingStatus
00544 {
00545     Header header;           ///< header.id 代表下发的 getGridMapFromAgv 指令 id
00546     GetGridMapStatus status; ///< 接口 getGridMapFromAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00547 };
00548
00549 /**
00550  * 接口 sendGridMapToAgv 的工作状态
00551  */
00552 struct SendGridMapWorkingStatus
00553 {
00554     Header header;            ///< header.id 代表下发的 sendGridMapToAgv 指令 id
00555     SendGridMapStatus status; ///< 接口 sendGridMapToAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00556 };
00557
00558 struct GetPngMapWorkingStatus
00559 {
00560     Header header;          ///< header.id 代表下发的 getBase64PngMapFromAgv 指令 id
00561     GetPngMapStatus status; ///< 接口 getBase64PngMapFromAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结
      束
00562 };
00563
00564 /**
00565  * 接口 sendBase64PngMapToAgv 的工作状态
00566  */
00567 struct SendPngMapWorkingStatus
00568 {
00569     Header header;           ///< header.id 代表下发的 sendBase64PngMapToAgv 指令 id
00570     SendPngMapStatus status; ///< 接口 sendBase64PngMapToAgv 的运行状态，NONE -接口未执行 RUNNING-执行中，FINISH-执行结
      束
00571 };
00572
00573 /**
00574  * 接口 setGridMapAllInfo 的工作状态
00575  */
00576 struct SetGridMapAllInfoWorkingStatus
00577 {
00578     Header header;                ///< header.id 代表下发的 setGridMapAllInfo 指令 id
00579     SetGridMapAllInfoStatus status; ///< 接口 setGridMapAllInfo 的运行状态
00580                                   ///< NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00581 };
00582
00583 /**
00584  * 接口 getGridMapAllInfo 的工作状态
00585  */
00586 struct GetGridMapAllInfoWorkingStatus
00587 {
00588     Header header;                ///< header.id 代表下发的 getGridMapAllInfo 指令 id
00589     GetGridMapAllInfoStatus status; ///< 接口 getGridMapAllInfo 的运行状态
00590                                   ///< NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00591 };
00592
00593 /**
00594  * 接口 previewPngMapFromAgv 的工作状态
00595  */
00596 struct PreviewPngMapWorkingStatus
00597 {
00598     Header header;              ///< header.id 代表下发的 previewPngMapFromAgv 指令 id
00599     PreviewPngMapStatus status; ///< 接口 previewPngMapFromAgv 的运行状态
00600                                 ///< NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00601 };
00602
00603 /**
00604  * 接口 autoAlignRailway 的工作状态
00605  */
00606 struct AutoAlignRailwayWorkingStatus
00607 {
00608     Header header;                ///< header.id 代表下发的 autoAlignRailway 指令 id
00609     AutoAlignRailwayStatus status; ///< 接口 autoAlignRailway 的运行状态
```

```
00610                                   ///< NONE -接口未执行 RUNNING-执行中，FINISH-执行结束
00611 };
00612
00613 /**
00614  * 异步接口运行状态
00615  * 暂时存在 14 个异步接口：saveMap，switchMap，changeRunningMode，relocation，sendBase64PngMapToAgv
00616  * getGridMapAllInfo，setGridMapAllInfo，sendGridMapToAgv，getGridMapFromAgv
00617  * getPngMapAllInfo，setPngMapAllInfo，getBase64PngMapFromAgv，previewPngMapFromAgv，autoAlignRailway
00618  */
00619 struct AsyncInterfaceResultStatus
00620 {
00621     Header header;                                      ///< agv 的 SN 码，agv 的名称，当前时间，地图的 id
00622     SaveMapWorkingStatus save_map_status;               ///< 保存地图（saveMap）接口的执行状态
00623     SwitchMapWorkingStatus switch_map_status;           ///< 切换地图（switchMap）接口的执行状态
00624     ChangeModeWorkingStatus change_running_mode_status; ///< 切换运行模式（changeRunningMode）接口的执行状态
00625     RelocationWorkingStatus relocation_status;          ///< 重定位（relocation）接口的执行状态
00626     GetGridMapWorkingStatus get_grid_map_status;        ///< 获取栅格地图（getGridMapFromAgv）接口的执行状态
00627     SendGridMapWorkingStatus send_grid_map_status;      ///< 发送栅格地图（sendGridMapToAgv）接口的执行状态
00628     GetPngMapWorkingStatus get_png_map_status;          ///< 获取 png 地图（getBase64PngMapFromAgv）接口的执行状态
00629     SendPngMapWorkingStatus send_png_map_status;        ///< 发送 png 地图（sendBase64PngMapToAgv）接口的执行状态
00630     SetPngMapAllInfoWorkingStatus set_png_all_status;   ///< 设置 png 地图全部信息（setPngMapAllInfo）接口的执行状态
00631     GetPngMapAllInfoWorkingStatus get_png_all_status;   ///< 获取 png 地图全部信息（getPngMapAllInfo）接口的执行状态
00632     SetGridMapAllInfoWorkingStatus set_grid_all_status; ///< 设置栅格地图全部信息（setGridMapAllInfo）接口的执行状态
00633     GetGridMapAllInfoWorkingStatus get_grid_all_status; ///< 获取栅格地图全部信息（getGridMapAllInfo）接口的执行状态
00634     PreviewPngMapWorkingStatus preview_png_map_status;  ///< 预览 png 地图（previewPngMapFromAgv）接口的执行状态
00635     AutoAlignRailwayWorkingStatus align_railway_status; ///< 自动上轨（autoAlignRailway）接口的执行状态
00636 };
00637
00638 /**
00639  * 站点
00640  */
00641 struct StationMark
00642 {
00643     Header header;    ///< 站点 id，站点名称，当前时间，站点所在地图的 id
00644     std::string type; ///< 普通站点是"ST"，充电站点“CP”
00645     Pose2d pose;      ///< 站点位姿，{x(m) y(m) theta(rad)}
00646 };
00647
00648 /**
00649  * 通过站点表示路径
00650  */
00651 struct PathStation
00652 {
00653     Header header;                      ///< 路径 id，路径名称，当前时间，路径所在地图的 id
00654     PathShape shape;                    ///< 路径的形状
00655     bool use_direction;                 ///< 生成站点时是否使用站点方向
00656     double max_speed;                   ///< agv 经过该路径时的最大速度
00657     std::string start_station_id;       ///< 路径起点站点 id
00658     std::string end_station_id;         ///< 路径终点站点 id
00659     std::vector<Point2d> control_points; ///< 路径的控制点，{{x,y},{}, ...}
00660 };
00661
00662 /**
00663  * 地图信息
00664  */
00665 struct MapInfo
00666 {
00667     Header header;            ///< 地图 id，地图名称，当前时间，map_id 为地图的唯一 id
00668     double resolution;        ///< 分辨率，每个正方形栅格/像素的宽度，单位 m
00669     uint32_t width;           ///< 地图宽度，多少个栅格/像素宽度
00670     uint32_t height;          ///< 地图高度，多少个栅格/像素高度
00671     Pose2d origin;            ///< {x(m) y(m) theta(rad)}
00672                               ///< 地图原点在真实地图下的坐标，指地图原点对应的实际位置坐标。
00673                               ///< 一张地图中，左下角为地图原点；
00674     MapFormat format;         ///< 地图格式
00675     std::vector<int8_t> data; ///< 地图数据信息
00676 };
00677
00678 /**
00679  *  @brief 栅格地图
00680  *  @details
00681  *  Header header;            // 地图 id，地图名称，当前时间，map_id 为地图的唯一 id\n
00682  *  double resolution;        // 分辨率，每个正方形栅格的宽度，单位 m\n
00683  *  uint32_t width;           // 地图宽度，多少个栅格宽度 \n
00684  *  uint32_t height;          // 地图高度，多少个栅格高度 \n
00685  *  Pose2d origin;            // {x(m) y(m) theta(rad)}
00686  *                            // 栅格地图原点在真实地图下的坐标，指栅格原点对应的实际位置坐标。
00687  *                            // 一张栅格地图中，左下角为栅格地图原点；\n
00688  *  MapFormat format;         // 地图格式，=MapFormat::OCCUPANCY_GRID\n
00689  *  std::vector<int8_t> data; // 地图数据信息；0 表示可通行，100 表示被占用，-1 表示未知
00690  */
00691 typedef MapInfo OccupancyGridMap;
00692
00693 /**
00694  *  @brief Base64 编码的 png 地图图片
00695  *  @details
00696  *  Header header;            // 地图 id，地图名称，当前时间，map_id 为地图的唯一 id\n
```

```
00697  *  double resolution;        // 分辨率，每个正方形像素表示的宽度，单位 m\n
00698  *  uint32_t width;           // 地图宽度，多少个像素宽度 \n
00699  *  uint32_t height;          // 地图高度，多少个像素高度 \n
00700  *  Pose2d origin;            // {x(m) y(m) theta(rad)}
00701  *                            // 地图原点在真实地图下的坐标，指地图原点对应的实际位置坐标。
00702  *                            // 一张地图中，左下角为地图原点；\n
00703  *  MapFormat format;         // 地图格式，=MapFormat::BASE64_PNG\n
00704  *  std::vector<int8_t> data; // 地图数据信息；Base64 编码的 png 地图图片，可以将 data 类型转换为 string 类型使用
00705  */
00706 typedef MapInfo Base64PngMap;
00707
00708 /**
00709  * 地图中的虚拟区域
00710  */
00711 struct MapVirtualArea
00712 {
00713     Header header;               ///< 虚拟区域的 id，虚拟区域的名称，当前时间，虚拟区域所在地图的 id
00714     MapVirtualAreaType type;     ///< 虚拟区域的类型
00715     MapVirtualAreaShape shape;   ///< 虚拟区域形状
00716     double width_speed;          ///< 虚拟墙的宽度/膨胀半径/减速区域的最大速度
00717     std::vector<Point2d> vertex; ///< 虚拟区域的顶点 {{x(m) y(m)},{}, ...}
00718                                  ///< 直线、多边形：按照顺时针顺序把顶点加入到 vertex 中
00719                                  ///< 圆弧：圆弧起点、圆弧上一点、圆弧终点
00720                                  ///< 圆形：圆上按顺序排序的任意不共线三个以上的点（包含 3 个）
00721 };
00722
00723 /**
00724  * 包含当前地图、当前地图上的站点、当前地图上的路径、当前地图上的虚拟区域信息
00725  */
00726 struct MapAllInfo
00727 {
00728     MapInfo map;                                ///< 当前地图信息
00729     std::vector<StationMark> stations;          ///< 当前地图上的站点信息
00730     std::vector<PathStation> paths;             ///< 当前地图上的路径信息
00731     std::vector<MapVirtualArea> virtual_areas; ///< 当前地图上的虚拟区域信息
00732 };
00733
00734 /**
00735  * IP 地址
00736  */
00737 struct IpAddressInfo
00738 {
00739     std::string name;         ///< 连接名称
00740     std::string type;         ///< 类型
00741     std::string device;       ///< 网卡名称
00742     std::string ipv4_ip;      ///< IPv4 地址及 CIDR 码，为空时无效
00743     std::string ipv4_gateway; ///< 网关
00744     std::string ipv4_method;  ///< 自动分配或者手动连接，对应字段 auto，manual
00745     std::string ipv4_dns;     ///< 第一个 DNS
00746     std::string ipv6_ip;      ///< IPv6 地址及 CIDR 码，为空时无效
00747     std::string ipv6_gateway;
00748 };
00749 /**
00750  * agv 到点后二次调整位置参数设置
00751  */
00752 struct AdjustParam
00753 {
00754     bool enable;         ///< 是否要二次调整 agv 位置，true-调整；false 不调整
00755     bool forward_flag;   ///< agv 调整方向，true，向车头方向调整，false，向车尾方向调整
00756     double max_distance; ///< agv 单次调整的最远距离
00757     double max_angle;    ///< agv 单次调整的最大角度
00758 };
00759 /**
00760  * 使 agv 导航到目标位置（控制信息）
00761  */
00762 struct NavGoalType
00763 {
00764     Header header;                       ///< 任务 id，任务名称，当前时间，地图 id
00765     NavType type;                        ///< 导航类型
00766     Pose2d target_pose;                  ///< 目标位姿 {x(m) y(m) theta(rad)}
00767                                          ///< 导航类型为自由导航到任意点时有效，其他无效
00768     std::vector<std::string> stations_id; ///< 导航经过的站点 id，最后一个站点为导航目标位置
00769                                          ///< 导航类型为自由导航到站点时，或路径导航到站点时有效，其他无效
00770     bool forward_flag;                   ///< agv 正向行驶或倒车行驶
00771     bool goal_end_rotate;                ///< 到达目标点后是否根据站点方向旋转，true：旋转
00772                                          ///< false：不旋转
00773     AdjustParam secondary_adjustment;    ///< 到达目标点后是否进行二次调整，贴合目标点位置，true：到点调整
00774                                          ///< false：到点不调整
00775     Speed max_vel;                       ///< 最大速度 v,w
00776 };
00777
00778 /**
00779  * @brief 导航状态信息  / 自动充电状态信息
00780  * @details
00781  * 导航模式时，获取导航状态信息 \n
00782  * 自动充电模式时，获取自动充电状态信息，type 与 current_trajectory 无效
00783  *  */
```

```
00784 struct NavInfo
00785 {
00786     Header header;    ///< 任务 id，任务名称，当前时间，地图 id
00787     NavStatus status; ///< 导航状态
00788     [[deprecated("Since 0.7.0: use nav_type instead.")]]
00789     NavType type;     ///< @if Chinese @deprecated 自 0.7.0 起废弃，请使用 nav_type 替代。 @endif @if English
      @deprecated Deprecated since 0.7.0. Use nav_type instead. @endif
00790     NavType nav_type;                       ///< 导航类型
00791     AutoChargingType auto_charging_type;    ///< 自动充电类型
00792     StationMark goal_station;               ///< agv 行驶路径目标站点
00793     std::vector<Point2d> current_trajectory; ///< 还没经过的路径点 {{x(m) y(m)},{}, ...}
00794 };
00795
00796 /**
00797  * @brief 自动充电信息
00798  * @since 0.7.0
00799  */
00800 struct AutoChargingCommand
00801 {
00802     Header header;                      ///< 任务 id，任务名称，当前时间，地图 id
00803     AutoChargingType auto_charging_type; ///< 自动充电类型
00804     std::string cp_station_id;          ///< 站点 id，上桩任务中，表示去往哪个充电站点进行充电；为空时前往最近充电站点
00805     std::string leave_to_station_id;    ///< 站点 id，下桩任务中，表示下桩导航到哪个站点；为空时导航到充电站点
00806     Speed max_speed;                    ///< 充电过程中的最大行驶速度
00807 };
00808
00809 /**
00810  * 配置要更新固件的模块及固件路径
00811  */
00812 struct FirmwareUpdateParam
00813 {
00814     FirmwareUpdateMode left_motor;   ///< 左电机固件更新信息配置
00815     FirmwareUpdateMode right_motor;  ///< 右电机固件更新信息配置
00816     FirmwareUpdateMode master_board; ///< 接口板固件更新信息配置
00817     std::string firmware_file_path;  ///< 代表固件包的路径 例如 "/root/update.firm"
00818 };
00819
00820 /**
00821  * 固件更新过程信息
00822  */
00823 struct FirmwareUpdateProcessInfo
00824 {
00825     std::string update_info; ///< 当前固件更新的步骤信息，failed 为升级失败，
00826                              ///< none 为无固件更新任务
00827     double update_progress;  ///< 固件更新进度信息，1.0 代表升级成功
00828 };
00829
00830 struct CalibrationProcessInfo
00831 {
00832     Header header;                 ///< 任务 id，任务名称，当前时间，地图 id
00833     CalibrationType type;          ///< 标定类型
00834     CalibrationStatus status;      ///< 标定状态
00835     uint32_t collected_data_count; ///< 采集到多少组数据
00836 };
00837
00838 /**
00839  * RTDE 菜单
00840  */
00841 struct RtdeRecipe
00842 {
00843     bool to_server;                    ///< 输入/输出
00844     int chanel;                        ///< 通道
00845     double frequency;                  ///< 更新频率
00846     int trigger;                       ///< 触发方式（该功能暂未实现）：0 - 周期；1 - 变化
00847     std::vector<std::string> segments; ///< 字段列表
00848 };
00849
00850 enum error_type
00851 {
00852     parse_error      = -32700, ///< 解析错误
00853     invalid_request  = -32600, ///< 无效请求
00854     method_not_found = -32601, ///< 方法未找到
00855     invalid_params   = -32602, ///< 无效参数
00856     internal_error   = -32603, ///< 内部错误
00857     server_error,              ///< 服务器错误
00858     invalid                    ///< 无效
00859 };
00860
00861 enum ExceptionCode
00862 {
00863     EC_DISCONNECTED     = -1,  ///< 断开连接
00864     EC_NOT_LOGINED      = -2,  ///< 未登录
00865     EC_INVAL_SOCKET     = -3,  ///< 无效套接字
00866     EC_REQUEST_BUSY     = -4,  ///< 请求繁忙
00867     EC_SEND_FAILED      = -5,  ///< 发送失败
00868     EC_RECV_TIMEOUT     = -6,  ///< 接收超时
00869     EC_RECV_ERROR       = -7,  ///< 接收错误
```

```
00870     EC_PARSE_ERROR      = -8,  ///< 解析错误
00871     EC_INVALID_REQUEST  = -9,  ///< 无效请求
00872     EC_METHOD_NOT_FOUND = -10, ///< 方法未找到
00873     EC_INVALID_PARAMS   = -11, ///< 无效参数
00874     EC_INTERNAL_ERROR   = -12, ///< 内部错误
00875     EC_SERVER_ERROR     = -13, ///< 服务器错误
00876     EC_INVALID          = -14  ///< 无效
00877 };
00878
00879 class AgvcException : public std::exception
00880 {
00881 public:
00882     AgvcException(int code, const std::string &prefix, const std::string &message) noexcept
00883         : code_(code), message_(prefix + "-" + message)
00884     {
00885     }
00886
00887     AgvcException(int code, const std::string &message) noexcept : code_(code), message_(message) {}
00888
00889     error_type type() const
00890     {
00891         if (code_ >= -32603 && code_ <= -32600) {
00892             return static_cast<error_type>(code_);
00893         } else if (code_ >= -32099 && code_ <= -32000) {
00894             return server_error;
00895         } else if (code_ == -32700) {
00896             return parse_error;
00897         }
00898         return invalid;
00899     }
00900
00901     int code() const { return code_; }
00902     const char *what() const noexcept override { return message_.c_str(); }
00903
00904 private:
00905     int code_;
00906     std::string message_;
00907 };
00908
00909 inline const char *returnValue2Str(int retval)
00910 {
00911     static const char *retval_str[] = {
00912 #define ENUM_ITEM(n, v, s) s,
00913         ENUM_AgvcErrorCodes_DECLARES
00914 #undef ENUM_ITEM
00915     };
00916
00917     enum agvc_index
00918     {
00919 #define ENUM_ITEM(n, v, s) n##_INDEX,
00920         ENUM_AgvcErrorCodes_DECLARES
00921 #undef ENUM_ITEM
00922     };
00923
00924     int index = -1;
00925
00926 #define ENUM_ITEM(n, v, s) \
00927     if (retval == v)       \
00928         index = n##_INDEX;
00929     ENUM_AgvcErrorCodes_DECLARES
00930 #undef ENUM_ITEM
00931
00932         if (index == -1)
00933     {
00934         index = AGVC_ERR_UNKOWN;
00935     }
00936
00937     return retval_str[(unsigned)index];
00938 }
00939
00940 } // namespace agvc_interface
00941
00942 #if defined ENABLE_JSON_TYPES
00943 #include "bindings/jsonrpc/json_types.h"
00944 #endif
00945
00946 #endif
```

Index