方法

方法原型:

```
bool Connect(const std::string &ip, uint16_t port);
```

方法描述:

连接到机器人服务.

方法参数:

参数名称	描述
ip	机器人服务的IP地址
port	机器人服务的端口

返回值:

成功:true,成功连接到机器人服务.

失败:false,通过LastError方法获取失败原因.

方法原型:

```
int ProcessEvent(int max_event_count = 0);
```

方法描述:

信号驱动函数(信号由事件驱动),此函数应该放在程序中的主循环中.

方法参数:

参数名称	描述
max_event_count	事件处理计数,默认0为处理所有事件

返回值:

被处理的事件个数.

方法原型:

```
ErrorCode LastError()const;
```

方法描述:

获取最后一次错误.

返回值:

错误码

方法原型:

```
bool RegisterMe(int64_t appid=0, const std::string &sercet=std::string());
```

方法描述(异步调用):

注册SDK,使用SDK,其他功能之前,须先注册.

方法参数:

参数名称	描述
appid	由行深分配
sercet	由行深分配

返回值:

成功:true,仅指示异步调用成功,具体结果参照信号返回.

失败:false,通过LastError方法获取失败原因.

相关信号:

sig_register_return

方法原型:

```
bool UnregisterMe();
```

方法描述(异步调用):

退出注册,SDK退出之前,须调用此方法.

返回值:

成功:true,仅指示异步调用成功,具体结果参照信号返回.

失败:false,通过LastError方法获取失败原因.

相关信号:

sig_unregister_return

方法原型:

```
        int QueryParameter();
```

方法描述(异步调用):

获取机器人参数设置

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_unregister_return

方法原型:

```
int QueryRoute();
```

方法描述(异步调用):

路线查询

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_query_route_return

方法原型:

```
int MappingControl(const MapperControl& param);
```

方法描述(异步调用):

建图控制

方法参数:

参见结构数据MapperControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_mapping_control_return

方法原型:

```
int QueryMap();
```

方法描述(异步调用):

获取地图

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_query_map_return

方法原型:

```
int SubscribeInfomation(const Subscribe& param);
```

方法描述(异步调用):

数据订阅,获取车辆相关信息.

方法参数:

参见结构数据Subscribe

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_subscribe_infomation_return

sig_vehicle_state

sig_Fuel_state

sig_switch_state

sig_task_state

sig_partrol_task_state

sig_global_pose

sig_arrive_station

sig_turn

sig_vehicle_start

sig_vehicle_stop

sig_obstacle

sig_fault

方法原型:

```
        int StartTask(const StationInfo& param);
```

方法描述(异步调用):

启动任务

方法参数:

参见结构数据StationInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_start_task_return

方法原型:

```
int StarPathTask(const PathInfo& param);
```

方法描述(异步调用):

启动路线任务

方法参数:

参见结构数据PathInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_start_path_task_return

方法原型:

```
int ControlTask(const TaskControl &param);
```

方法描述(异步调用):

任务控制相关

方法参数:

参见结构数据TaskControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_control_task_return

方法原型:

```
int Navigate(const NavigationPoint &param);
```

方法描述(异步调用):

导航到任意点

方法参数:

参见结构数据NavigationPoint

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_navigate_return

方法原型:

```
int QueryTaskObject(const GetTaskObject &param);
```

方法描述(异步调用):

获取列表

方法参数:

参见结构数据GetTaskObject

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_qto_path_info_return

sig_qto_station_info_return

sig_qto_patro_info_return

sig_qto_group_info_return

sig_qto_map_info_return

方法原型:

```
int SetStation(const StationInfo &param);
```

方法描述(异步调用):

设置停靠点

方法参数:

参见结构数据StationInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_set_station_return

方法原型:

```
int SetParameter(const StationInfo &param);
```

方法描述(异步调用):

设置参数

方法参数:

参见结构数据StationInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_set_parameter_return

方法原型:

```
int ModifyPathName(const PathInfo &param);
```

方法描述(异步调用):

修改路线名称

方法参数:

参见结构数据PathInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_modify_path_name_return

方法原型:

```
int ControlLocalisation(const LocalisationControl &param);
```

方法描述(异步调用):

定位空置相关

方法参数:

参见结构数据LocalisationControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_control_localisation_return

方法原型:

```
int QueryLocalisationPoint();
```

方法描述(异步调用):

拉去定位点列表

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_query_localisation_point_return

方法原型:

```
int QueryLocalisationData(const LocalisationControl &param);
```

方法描述(异步调用):

拉去定位数据

方法参数:

参见结构数据LocalisationControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_query_localisation_data_return

方法原型:

```
int SetLocalisationResult(const LocalisationResult &param);
```

方法描述(异步调用):

设置定位结果

方法参数:

参见结构数据LocalisationResult

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_set_localisation_result_return

方法原型:

```
int QueryGlobalMap();
```

方法描述(异步调用):

获取全局底图

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_query_global_map_return

方法原型:

```
int ControlRouteRecording(const RouteRecordControl &param);
```

方法描述(异步调用):

路径采集控制

方法参数:

参见结构数据RouteRecordControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_control_route_recording_return

方法原型:

```
int ControlSynchronization (const SyncControl &param);
```

方法描述(异步调用):

数据同步控制

方法参数:

参见结构数据SyncControl

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_control_synchronization_return

方法原型:

```
int ApplyMap(const MapInfo &param);
```

方法描述(异步调用):

使用地图

方法参数:

参见结构数据MapInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_apply_map_return

方法原型:

```
int UpdateMap(const MapInfo &param);
```

方法描述(异步调用):

方法参数:

更新地图

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_update_map__return

方法原型:

```
int DeleteMap(const MapInfo &param, bool all = false);
```

方法描述(异步调用):

删除地图

方法参数:

参见结构数据MapInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号:

sig_delete_map_return

方法原型:

```
int StartPointTask(const GridPoint &param);
```

方法描述(异步调用):

启动点任务

方法参数:

参见结构数据GridPoint

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_start_point_task_return

方法原型:

```
int StartPointListTask(const GridPointList &param);
```

方法描述(异步调用):

启动点列任务

方法参数:

参见结构数据GridPointList

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_start_point_list_task_return

方法原型:

```
int QueryMappingState();
```

方法描述(异步调用):

获取建图进度状态

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_query_mapping_state_return

方法原型:

```
int QueryLocalisationState();
```

方法描述(异步调用):

获取定位进度状态

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_query_localisation_state_return

方法原型:

```
int QueryLocalisationResult();
```

方法描述(异步调用):

获取定位结果

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_query_localisation_result_return

方法原型:

```
int QueryRouteRecordingState();
```

方法描述(异步调用):

获取路径采集进度状态

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_query_route_recording_state_return

方法原型:

```
int TransferMap(const MapRequest& param);
```

方法描述(异步调用):

地图上传下载

方法参数:

参见结构数据MapRequest

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_transfer_map_return

方法原型:

```
int ApplySite(const SiteInfo& param);
```

方法描述(异步调用):

使用站点

方法参数:

参见结构数据SiteInfo

返回值:

成功:>=0,调用序列号,仅指示异步调用成功,具体结果参照信号返回.

失败:-1,通过LastError方法获取失败原因.

相关信号

sig_apply_site_return

信号

信号原型:

```
signal0<> sig_disconnect
```

信号描述:

机器人服务与SDK失去连接的时候,该信号被触发.

信号原型:

```
signal1<const SimpleResponse*> sig_unregister_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_subscribe_infomation_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_mapping_control_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_start_task_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_control_task_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_start_path_task_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_navigate_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_set_station_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_set_parameter_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_modify_path_name_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_control_localisation_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_set_localisation_result_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_control_route_recording_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_control_synchronization_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_apply_map_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_update_map__return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_delete_map_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_start_point_task_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_start_point_list_task_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const SimpleResponse*> sig_transfer_map_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1 <const VehicleState*> sig_vehicle_state;
```

信号参数:

参考数结构数据**VehicleState**.

信号原型:

```
signal1<const Fuel*> sig_Fuel_state;
```

信号参数:

参考数结构数据**Fuel**.

信号原型:

```
signal1<const SwitchControl*> sig_switch_state;
```

信号参数:

参考数结构数据**SwitchControl**.

信号原型:

```
signal1<const TaskState*> sig_task_state;
```

信号参数:

参考数结构数据**TaskState**.

信号原型:

```
signal1<const PatrolTaskState*> sig_partrol_task_state;
```

信号参数:

参考数结构数据**PatrolTaskState**.

信号原型:

```
signal1<const GlobalPose*> sig_global_pose;
```

信号参数:

参考数结构数据**GlobalPose**.

信号原型:

```
signal1<const ArriveStation*> sig_arrive_station;
```

信号参数:

参考数结构数据**ArriveStation**.

信号原型:

```
signal1<const Turn*> sig_turn;
```

信号参数:

参考数结构数据**Turn**.

信号原型:

```
signal1<const VehicleStart*> sig_vehicle_start;
```

信号参数:

参考数结构数据**VehicleStart**.

信号原型:

```
signal1<const VehicleStop*> sig_vehicle_stop;
```

信号参数:

参考数结构数据**VehicleStop**.

信号原型:

```
signal1<const Obstacle*> sig_obstacle;
```

信号参数:

参考数结构数据**Obstacle**.

信号原型:

```
signal1<const Fault*> sig_fault;
```

信号参数:

参考数结构数据**Fault**.

信号原型:

```
signal1<const TaskRoute*> sig_query_route_return;
```

信号参数:

参考数结构数据**TaskRoute**.

信号原型:

```
signal1<const PathInfoResponse*> sig_qto_path_info_return;
```

信号参数:

参考数结构数据**PathInfoResponse**.

信号原型:

```
signal1<const StationInfoResponse*> sig_qto_station_info_return;
```

信号参数:

参考数结构数据**StationInfoResponse**.

信号原型:

```
signal1<const PatrolInfoResponse*> sig_qto_patro_info_return;
```

信号参数:

参考数结构数据**PatrolInfoResponse**.

信号原型:

```
signal1<const GroupInfoResponse*> sig_qto_group_info_return;
```

信号参数:

参考数结构数据**GroupInfoResponse**.

信号原型:

```
signal1<const MapInfoResponse*> sig_qto_map_info_return;
```

信号参数:

参考数结构数据**MapInfoResponse**.

信号原型:

```
signal1<const SiteInfoResponse*> sig_qto_site_info_return;
```

信号参数:

参考数结构数据**SiteInfoResponse**.

信号原型:

```
signal1<const ParameterSetting*> sig_query_parameter_return;
```

信号参数:

参考数结构数据SimpleResponse.

信号原型:

```
signal1<const StationInfoResponse*> sig_query_localisation_point_return;
```

信号参数:

参考数结构数据**StationInfoResponse**.

信号原型:

```
signal1<const LocalisationResponse*> sig_query_localisation_data_return;
```

信号参数:

参考数结构数据**LocalisationResponse**.

信号原型:

```
signal1<const BinaryData*> sig_query_global_map_return;
```

信号参数:

参考数结构数据**BinaryData**.

信号原型:

```
signal1<const RouteRecordResponse*> sig_route_record_result;
```

信号参数:

参考数结构数据**RouteRecordResponse**.

信号原型:

```
signal1<const SyncProgress*> sig_sync_progress;
```

信号参数:

参考数结构数据**SyncProgress**.

信号原型:

```
signal1<const SyncState*> sig_sync_state;
```

信号参数:

参考数结构数据**SyncState**.

信号原型:

```
signal1<const MapperResponse*> sig_query_map_return;
```

信号参数:

参考数结构数据**MapperResponse**.

信号原型:

```
signal1<const MapperTackPoint*> sig_track_point;
```

信号参数:

参考数结构数据**MapperTackPoint**.

信号原型:

```
signal1<const MapperResponse*> sig_tile_map;
```

信号参数:

参考数结构数据**MapperResponse**.

信号原型:

```
signal1<const MapperCurrent*> sig_query_mapping_state_return;
```

信号参数:

参考数结构数据**MapperCurrent**.

信号原型:

```
signal1<const LocalisationCurrent*> sig_query_localisation_state_return;
```

信号参数:

参考数结构数据**LocalisationCurrent**.

信号原型:

```
signal1<const LocalisationResult*> sig_query_localisation_result_return;
```

信号参数:

参考数结构数据**LocalisationResult**.

信号原型:

```
signal1<const RouteRecordCurrent*>
sig_query_route_recording_state_return;
```

信号参数:

参考数结构数据**RouteRecordCurrent**.

信号原型:

```
signal3<int, const std::string&, const std::string&> sig_register_return;
```

信号参数:

参考数结构数据**MapperCurrent**.

信号原型:

```
signal1<const SimpleResponse*> sig_apply_site_return;
```

信号参数:

参考数结构数据SimpleResponse.

结构数据

```
struct SimpleResponse
{
 //调用序列号
 int32_t call_id;
 //应答结果
 int32_t result;
 //应答消息
 std::string message;
};

struct StationInfo
{
 string station_id;
 string station_name;
 //经度
 double longitude;
 //维度
 double latitude;
 //方向
 double azimuth;

 enum StationType
 {
 STATION =0,
 CHARGE = 1,
 PATROL = 2,
 GRID = 10000,
 };
 StationType type;
};

struct TaskConfig
{
```

```
 enum SpeedLevel
 {
 NORMAL = 0,
 LOW = 1,
 HIGH = 2,
 };
 //速度档位
 SpeedLevel speed_level;

 enum ObstacleType
 {
 //避障
 OBSTACLEAVOID = 0,
 //停障
 OBSTACLESTOP = 1,
 };
 ObstacleType obstacle_type;
};

struct GridInfo
{
 int32_t grid_height;
 int32_t grid_width;
 int32_t origin_x;
 int32_t origin_y;
 int32_t resolution;
};

struct MapperControl
{
 enum ControlType
 {
 //开始定位
 START_GMAPPER = 0,
 //结束定位
 STOP_MAPPER = 1,
 //保存定位结果
 SAVE_MAPPER = 2,
 //不保存定位结果
 DONOT_SAVE_MAPPER = 3,
```

```
 GET_TILE_MAP = 4, // not support
 //开始增量建图
 START_INCREASE_MAPPER = 5,
 //生成地图
 GENERATE_MAP = 6,
 //增加闭环点
 ADD_CLOSURE_POINT = 7,
 //匹配闭环
 MATCH_CLOSURE_POINT = 8,
 //暂停定位
 PAUSE_MAPPERR = 9,
 //恢复定位
 RESUME_MAPPER = 10,
 };
 ControlType control_type;
};

struct Subscribe
{
 enum SubscribeType
 {
 subscribe = 0,//订阅
 unSubscribe=1//取消订阅
 };
 //订阅的主题
 int32_t topic;
  SubscribeType subscribe_type;
  //数据发送周期 单位ms
  int32_t cycle;
};

struct VehicleState
{
  enum Shift
  {
  R = 0,
  P = 1,
  N = 2,
  D = 3,
  };
```

```
114
115  enum DriveMode
116  {
117  Standby = 0,
118  Manual = 1,
119  Auto = 2,
120  };
121
122  //当前时间
123  int64_t date_time;
124  //车辆定位经度
125  double longitude;
126  //车辆定位纬度
127  double latitude;
128  //车辆所在海拔高度
129  double altitude;
130  //前轮摆角度
131  double steer_angle;
132  //方向角（正东逆时针）
133  double azimuth;
134  //档位 RPND
135  Shift shift;
136  //油门开度（0-100）%
137  double throttle;
138  //刹车力度（0-100）%
139  double brake;
140  //行驶速度 m/s
141  double speed;
142  //驾驶模式 待机 手动 自动
143  DriveMode drive_mode;
144 };
145
146 struct Fuel
147 {
148  //剩余燃料
149  double residual_fuel;
150  //电池电压
151  double voltage;
152  //电流
153  double ammeter;
```

```
 //电池温度
 double temperature;
 //续航里程
 double endurance;
 bool is_charging;
 //累计里程
 double odometer;
};

struct SwitchControl
{
 enum Switch
 {
 ON = 0,
 OFF = 1,
 };

 enum SwitchType
 {
 WARNINGLAMP=0,
 MAINLAMP=1,
 SUBLAMP=2,
 FOGLAMP=3,
 WHISTLE=4,
 LEFTTURNLIGHT=5,
 RIGHTTURNLIGHT=6,
 };
 Switch switch_;
 int64_t delayed;
 SwitchType switch_type;
};

struct TaskState
{
 int64_t start_time;
 int32_t point_count;
 int32_t current_point_index;
 //起始站点
 StationInfo start_station;
 //结束站点
```

```
 StationInfo end_station;
};

struct PatrolTaskState
{
 string name; // patroltask name ongoing
 string id; // patroltask id ongoing
 int32_t task_num;
 int32_t current_task_index; // start from 0
 int32_t loop_num = 5;
 int32_t current_loop_index; // start form 0
 TaskConfig config;
 enum TaskStatus
 {
 STANDBY = 0,//有任务，待接收执行命令
 RUNNING = 1, //正在执行
 PAUSING = 2,//暂停中
 STOPPED = 3, //已停止
 };
 TaskStatus status;
};

struct EulerAngle
{
 double roll;
 double pitch;
 double yaw;
};

struct GlobalPose
{
 //时间戳
 int64_t date_time;

 enum GPSState{
 POWERON=0,// 上电
 INIT=1,// 初始化
 SUCCESS=2,// 成功
 };
 GPSState gps_state;
```

```
 int32_t gps_satellite_num;
 int32_t gps_week;
 int64_t gps_millisecond;
 enum PositionStatus
 {
 NONE=0,//未定位
 SINGLE=1,// 单点定位
 PSRDIFF=2,// 差分
 RTKFLOAT=3,// 浮点定位
 RTKFIX=4,// 差分定位
 UNKNOWN=5,// 未知
 };
 PositionStatus position_status;
 EulerAngle gps_angle;
 struct GPSVelocity
 {
 double north_velocity;
 double east_velocity;
 double up_velocity;
 };
 GPSVelocity gps_velocity;
 double lng;
 double lat;
 double altitude;

 EulerAngle gps_angle_dev;
 double lng_dev;
 double lat_dev;
 double gauss_x;
 double gauss_y;
};

struct ArriveStation
{
 //到达时间
 int64_t date_time;
 //到达站点的信息
 StationInfo station;
};

```

```
274
275 struct MapInfo
276 {
277  string id;
278  string name;
279  string create_time;
280  GridInfo grid_info;
281 };
282
283 struct Point3D
284 {
285  double x;
286  double y;
287  double z;
288 };
289
290 struct Turn
291 {
292  enum Direction
293  {
294  LEFT=0,
295  RIGHT=1,
296  };
297  //时间
298  int64_t date_time;
299  //方向
300  Direction direction;
301  //经度
302  double longitude;
303  //维度
304  double latitude;
305  //方向
306  double azimuth;
307 };
308
309 struct VehicleStart
310 {
311  int64_t date_time;
312  //经度
313  double longitude;
```

```
 //维度
 double latitude;
 //方向
 double azimuth;
};

struct VehicleStop
{
 //时间
 int64_t date_time;
 //经度
 double longitude;
 //维度
 double latitude;
 //方向
 double azimuth;
};

struct Obstacle
{
 int64_t date_time;
 double longitude;
 double latitude;
 double azimuth;

 struct ObstacleInfo
 {
 enum ObstacleType
 {
 PEDESTRIAN=0,
 VEHICLE=1,
 };
 ObstacleType obstacle_type;
 double x;
 double y;
 double z;
 };
 std::vector<ObstacleInfo> obstacles;
};

```

```
354
355 struct Fault
356 {
357  enum FaultModul
358  {
359  SENSOR=0,//传感器
360  CHASSIS=1,//底盘
361  APPLICATION=2,//应用
362  };
363
364  enum FaultType{
365  GEN=0,//产生
366  RELIEVE=1,//解除
367  };
368
369  enum FaultLevel
370  {
371  DEBUG=0,//调试
372  INFO=1,//消息
373  NOTICE = 2,
374  WARN=3,//警告
375  BUG=4,
376  ERROR=5,
377  FAULT=6,//故障
378  FAILURE=7,//致命
379  };
380
381  int64_t date_time;
382  string fault_code;
383  string fault_message;
384  FaultModul fault_modul;
385  FaultType fault_type;
386  FaultLevel fault_leve;
387  std::vector<uint8_t> fault_data;
388
389 };
390
391 struct TaskControl
392 {
393  enum ControlType
```

```
 {
 RUN = 0,
 SUSPEND = 1,
 CANCEL = 2,
 RESUME = 3,
 };

 ControlType control_type;
};

struct TaskRoute
{
 int64_t start_time;
 int64_t arrival_time;
 struct Point
 {
 double longitude;

 double latitude;

 double azimuth;
 };
 //当前位置
 Point current_point;
 //路线点集合
 std::vector<Point> route_points;
 //起始站点
 StationInfo start_station;
 //结束站点
 StationInfo end_station;
};

struct GPSLocation
{
 double longitude;
 double latitude;
 double azimuth;
 double altitude;
};

```

```
434 struct PathInfo
435 {
436  string path_id;
437  string path_name;
438  int32_t point_num;
439  GPSLocation start_point;
440  GPSLocation end_point;
441 };
442
443 struct PatrolInfo
444 {
445  string task_name;
446  string task_id;
447
448  enum TaskType
449  {
450  STATION_TASK = 0,
451  PATH_TASK = 1,
452  };
453  TaskType task_type;
454
455  struct
456  {
457  StationInfo station;
458  PathInfo path;
459  }task;
460
461  struct Date
462  {
463  // Year of date. Must be from 1 to 9999, or 0 if specifying a date wit
out
464  // a year.
465  int32_t year;
466  // Month of year. Must be from 1 to 12, or 0 if specifying a year witho
ut a
467  // month and day.
468  int32_t month;
469  // Day of month. Must be from 1 to 31 and valid for the year and month,
or 0
470  // if specifying a year by itself or a year and month where the day is
not
```

```
471  // significant.
472  int32_t day;
473  int32_t hour;
474  int32_t min;
475  };
476  Date date;
477  enum LoopType
478  {
479  CYCLE = 0,
480  ROUND = 1, // go there and back
481  };
482  LoopType loop_type;
483  int32_t loop_time;
484  TaskConfig config;
485 };
486
487 struct GroupInfo
488 {
489  //任务名
490  string task_name;
491  string task_id;
492  struct SubTask
493  {
494  enum TaskType
495  {
496  //停靠点任务
497  STATION_TASK = 0,
498  // 路线任务
499  PATH_TASK = 1,
500  };
501  TaskType task_type;
502  struct
503  {
504  StationInfo station;
505  PathInfo path;
506  } task;
507  // 任务配置
508  TaskConfig config;
509  //等待时间
510  int32_t wait_time; // wait time in min
```

```
511   };
512   std::vector<SubTask> sub_task;
513
514   struct Date
515   {
516   // Year of date. Must be from 1 to 9999, or 0 if specifying a date with
out
517   // a year.
518   int32_t year;
519   // Month of year. Must be from 1 to 12, or 0 if specifying a year witho
ut a
520   // month and day.
521   int32_t month;
522   // Day of month. Must be from 1 to 31 and valid for the year and month,
or 0
523   // if specifying a year by itself or a year and month where the day is
not
524   // significant.
525   int32_t day;
526   int32_t hour;
527   int32_t min;
528   };
529   //起始时间
530   Date date;
531   //循环次数
532   int32_t loop_time;
533   //持续时间
534   double last_hours;
535   //结束时间
536   Date end_time;
537  };
538
539  struct SiteInfo
540  {
541   string id;
542   //站点名字
543   string name;
544   //条带
545   int32_t zone;
546  };
547
```

```
548 struct GetTaskObject
549 {
550  enum TaskObjectType
551  {
552  STATION_INFO = 0, //停靠点
553  PATH_INFO = 1, //路径
554  MAP_POINT_INFO = 2, // not used
555  PATROL_INFO = 3, //巡逻任务
556  GROUP_INFO = 4, // 组任务
557  MAP_INFO = 5, // 地图信息
558  SITE_INFO = 6,// 站点信息
559  };
560  TaskObjectType object_type;
561 };
562
563 struct StationInfoResponse
564 {
565  std::vector<StationInfo> station;
566 };
567
568 struct PathInfoResponse
569 {
570  std::vector<PathInfo> path;
571 };
572
573 struct MapInfoResponse
574 {
575  std::vector<MapInfo> map;
576 };
577
578
579 struct PatrolInfoResponse
580 {
581  string name;
582  string id;
583  std::vector<PatrolInfo> patrol;
584 };
585
586 struct GroupInfoResponse
587 {
```

```
 string name;
 string id;
 std::vector<GroupInfo> patrol;
};

struct SiteInfoResponse
{
 std::vector<SiteInfo> site;
};

struct Pose2D
{
 double x;
 double y;
 double raw;
};

struct NavigationPoint
{
 Pose2D pose; //导航到任一点
 GPSLocation GPS_location;
};

struct ParameterSetting
{
 enum SpeedLevel
 {
 NORMAL = 0, //中档
 LOW = 1, //低档
 HIGH = 2, //高档
 };

 struct SpeedSetting
 {
 SpeedLevel level; //速度等级
 double speed; //速度值
 };

 std::vector<SpeedSetting> speed_setting;
};
```

```
struct LocalisationControl
{
 enum ControlType
 {
 LOCAL_BY_GPS = 0,//用GPS初始化定位
 LOCAL_BY_STATION = 1,//用站点初始化定位（需要station)
 LOCAL_MANUAL = 2,//手动定位
 LOCAL_CHECK = 3, //检查定位结果
 LOCAL_MANUAL_NO_GPS = 4,//无GPS时的手动定位（需要station)
 LOCAL_PAUSE = 5,//暂停定位
 LOCAL_RESUME = 6,//恢复定位
 LOCAL_STOP = 7,//停止定位
 };
 ControlType control_type;
 StationInfo station;
};

struct LocalisationResponse
{
 Pose2D pos;

 struct Range
 {
 double min_x;
 double min_y;
 double max_x;
 double max_y;
 int32_t grid_size;
 };
 Range range;
 std::vector<uint8_t> point_cloud; //点云数据
 std::vector<uint8_t> map_body; //地图数据
 int32_t quality; //定位质量（0~100）
};

struct LocalisationResult
{
 bool result; //定位结果 true 好 false 不好
 Pose2D pose;
};
```

```
669
670 struct BinaryData
671 {
672  std::vector<uint8_t> binary;
673 };
674
675 struct RouteRecordControl
676 {
677  enum ControlType
678  {
679  START_RECORD = 0,
680  STOP_RECORD = 1,
681  SAVE_RECORD = 2,
682  PAUSE_RECORD = 3,
683  RESUME_RECORD = 4,
684  };
685  ControlType control_type;
686 };
687
688 struct RouteRecordResponse
689 {
690  int32_t index;
691  Pose2D pose;
692  GPSLocation GPS_location;
693 };
694
695 struct SyncControl
696 {
697  enum ControlType
698  {
699  UPLOAD = 0,
700  DOWNLOAD = 1,
701  CANCEL = 2,
702  };
703
704  ControlType control_type;
705
706  enum DataType
707  {
708  //矢量图
```

```
 VECTOR = 0,
 //瓦片图
 MAP_TILE = 1,
 HDMAP = 2,
 MAP_3D = 3,
 MAP_2D = 4,
 //轨迹文件
 POSE = 5,
 //路网
 PATH_NET = 6,
 };

 std::vector<DataType> data_types;
};

struct SyncProgress
{
 struct DataProgress
 {
 double total;
 double current;
 };

 std::vector<DataProgress> progress;
 double total;
 double current;
};

struct SyncState
{
 enum State
 {
 ERROR = 0,
 COMPLETE = 1,
 };
 State state;
 string error_code;
 string error_msg;
};

```

```
749 struct MapperResponse
750 {
751  struct MapRange
752  {
753  double min_x;
754  double min_y;
755  double max_x;
756  double max_y;
757  int32_t grid_size;
758  };
759 
760  MapRange range;
761  std::vector<uint8_t> map_body;
762 };
763 
764 struct MapperTackPoint
765 {
766  int32_t index;
767  Pose2D pose;
768  GPSLocation GPS_location;
769 };
770 
771 struct MapperProgress
772 {
773  enum ControlType
774  {
775  UNKOWN = 0,
776  GENERATE_MAP = 6,
777  };
778 
779  ControlType type;
780 
781  enum Result
782  {
783  FAILED = 0,
784  SUCCESS = 1,
785  ONGOING = 2,
786  };
787  Result result;
788  string result_msg;
```

```
 int32_t progess; // 0~100
};

struct GridPoint
{
 string id;
 string name;
 string map_name;
 Point3D position;
 EulerAngle angle;
};

struct GridPointList
{
 std::vector<GridPoint> point;
};

struct MapperCurrent
{
 enum MapStatus
 {
 NORMAL = 0, //默认状态
 START_MAP = 1, //开始建图
 STOP_MAP = 2, //结束建图
 GENERATE_MAP = 3, //生成地图
 PAUSE_MAP = 4, //暂停建图
 RESUME_MAP = 5, //恢复建图
 };
 MapStatus status;
};

struct LocalisationCurrent
{
 enum LocalisationStatus
 {
 NORMAL = 0, //默认状态
 START_LOCAL = 1, //开始初始化定位
 STOP_LOCAL = 2, //停止初始化定位
 PAUSE_LOCAL = 3, //暂停初始化定位
 RESUME_LOCAL = 4, //恢复初始化定位
```

```
 };
 LocalisationStatus status;
};

struct RouteRecordCurrent
{
 enum RouteRecordStatus
 {
 NORMAL = 0, //默认状态
 START = 1,//开始
 STOP = 2,//停止
 PAUSE = 3,//暂停
 RESUME = 4, //恢复
 };
 RouteRecordStatus status;
};

struct MapRequest
{
 enum ControlType
 {
 DOWNLOAD = 0,
 UPLOAD = 1,
 };
 ControlType control_type;
 string address;
};
```