EGO API 接口文档

深圳玩智商科技有限公司

版本修改历史：

日期	版本号	作者	说明
2018-12-13	V1.0.1.2	李光永	初始发布
2018-12-22	V1.0.1.3	李光永	状态控制　增加 7，扩展建图。
2018-12-24	V1.0.1.4	李光永	增加 PUB 超声点云。
2018-12-28	V1.0.1.5	李光永	增加用户更改密码 API
2018-12-29	V1.0.1.6	李光永	增加录制数据包，获取资源/删除资源中增加数据包类型
2019-01-03	V1.0.1.7	李光永	增加下载录制数据包 API

注 1：地图名称，任务名称等中不能使用字符，如 -, # & [](){}。即可以使用字母，数字以及_ 。文件中，接口传递的参数中，使用字符来做分隔符。

注 2：程序中断重启后可以通过获取状态与机器人进行同步。

注 3：选择地图，机器人也会加载此地图。同时也需要设置机器人初始位置。

注 4：当版本为 VER=1 时（iVersion=1），ON_EAI_OccupancyGridMap 接到收到数据为地图名称，格式是 PNG，显示图片之前，请判断图片是否为 PNG 格式，读取完后请删除。

注 5：栅格坐标是图片左下角为圆点。

1. 初始化 DLL

```
typedef bool (__stdcall *InitDll)(HWND parent, int iWorkMode, int iBeatPacketMs);
{
```

Parent：主窗口句柄。

iWorkMode：工作模式，0,发送消息到句柄;1，回调函数发送消息。

iBeatPacketMs：心跳包发送间隔,200ms 到 5000ms。

}

2. 释放 DLL

```
typedef bool(__stdcall *ReleaseDll)(void);
```

3. 回调函数及注册回调函数

```
typedef void(*CallbackFun)(int  iMsgID, DWORD dwDataLen, char *pBuff);
typedef void(*SetCallBackFun)(CallbackFun callbackfun);
```

{

iMsgID：消息 ID。

dwDataLen：数据长度。

pBuff：数据缓冲区指针。

}

注：自定义消息 *WPARAM* 为 dwDataLen, *LPARAM* 为 pBuff

消息 ID:

```
#define ON_ACCEPT                    WM_USER + 1202
#define ON_EAI_PointCloud            WM_USER + 1203//激光
#define ON_EAI_OccupancyGridMap      WM_USER + 1204//地图
#define ON_EAI_Pose                  WM_USER + 1205//机器人位置
#define ON_EAI_Path                  WM_USER + 1206////局部路径
#define ON_EAI_JOY                   WM_USER + 1215//手柄控制消息
#define ON_EAI_GLOBAL_Path           WM_USER + 1216//全局路径
#define ON_EAI_STAT_INFO             WM_USER + 1217//状态信息
#define ON_EAI_DEBUG_POINT           WM_USER + 1218//调试点信息
```

```
#define ON_EAI_Battery                  WM_USER + 1221//电池信息
#define ON_EAI_Emergency                WM_USER + 1222//急停信息
#define ON_EAI_ErrorCode                WM_USER + 1223//错误信息
#define ON_EAI_BOT_PARENT               WM_USER + 1224//机器人类消息
#define ON_EAI_InitStatus               WM_USER + 1225/VER1 机器人初始化状态消息  int 0 完成，5 未完成
#define ON_EAI_Sonar                    WM_USER + 1226//超声 和激光数据一样
#define ON_EAI_DataBag                  WM_USER + 1227//数据包

/////名称长度

#define LEN_NAME_MAP                    40

#define LEN_NAME_MISSION                80

#define LEN_NAME_TARGETS                120

#define LEN_NAME_VIRPOINTS              80

#define LEN_NAME_PATHS                  120

#define LEN_NAME_DATABAG                40

////电池

typedef struct

{

    UINT64                  timestamp ;

    int                     iStatus;

    int                     iHealth;

    int                     iTech;

    float                   fVoltage;

    float                   fCurrent;

    float                   fCharge;
```

```
    float                   fCapacity;
    float                   fDesign_capacity;
    float                   fPercentage;
    BOOL                    bPresent;
}tsDATA_Battery;


typedef struct
{
    float                   fX;
    float                   fY;
    float                   fPhi;
}tsDATA_Pose2D;

typedef struct
{
    float                   fX;
    float                   fY;
}tsDATA_Point2D;

typedef struct
```

```
{
    int                     iX;
    int                     iY;
}tsDATA_Point2D_B;;
////VER=1，点云 栅格化的坐标

/////点云
typedef struct
{
    UINT64                  u64Timestamp;
    UINT32                  u32ID;
    UINT                    u32Number;
    tsDATA_Pose2D           sOriginPose2d;
    tsDATA_Point2D          *psDataPoint2d;
}tsDATA_PointCloud;

/////地图
typedef struct
{
    UINT64                  u64Timestamp;
    float                   fResolution;
    UINT                    u32Width ;
```

```
    UINT                    u32Heigth;
    tsDATA_Pose2D           sOriginPose2d;
    char                    *p8Data;
}tsDATA_OccupancyGridMap;
////位置
typedef struct
{
    UINT64                  u64Timestamp;
    tsDATA_Pose2D           sPose2d;
}tsDATA_Pose;


////路径
typedef struct
{
    UINT64                  u64Timestamp;
    char                    chName[LEN_NAME_PATHS];
    UINT32                  u32Pose2dNum;
    tsDATA_Pose2D           *psPoses2d;
}tsDATA_Path;
```

///状态

typedef struct

{

UINT64 u64Timestamp;

char chMapName[LEN_NAME_MAP];

char chMissionName[LEN_NAME_MISSION];

UINT32 u32Stat;

UINT32 u32Action_id;

UINT32 u32Action_type;

UINT32 u32Action_state;

}tsDATA_Stat_Info;

4. 用户登录

```
typedef int(__stdcall *EAILogin)(char *chUserName, char *chPassword);
```

{

chUserName：用户名，最大 20 个字节。

chPassword：密码，最大 20 个字节。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

5. 更改密码

```
typedef int(__stdcall *ChangePwd)(char *chUserName, char *chPassword);
```

{

chUserName：用户名，最大 20 个字节。

chPassword：密码，最大 20 个字节。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

6. 获取地图列表

```
typedef int(__stdcall *GetMapList)(void);
```

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回成功后，读取文件：SourceList\EAIMap.eai

示例：地图名,时间戳,关联任务 1#关联任务 2#

ego,2018-09-07 14:42:03.586,mission1#

7. 获取任务列表

```
typedef int(__stdcall *GetMissionList)(void);
```

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回成功后，读取文件：SourceList\EAIMission.eai

示例：任务名,时间戳,地图名#

transport,2018-09-10 09:54:58.572,ego#

8. 获取数据包列表

```
typedef int(__stdcall *GetDataBagList)(void);
```

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回成功后，读取文件：SourceList\EAIDataBag.eai

示例：数据包名,时间戳,

transport,2018-09-10 09:54:58.572,

9. 获取地图

```
typedef int(__stdcall *GetMapByName)(char *chMapName);
```

{

chMapName：地图名称。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回成功后，读取文件：OccupancyGridMap\chMapName.eai

地图文件解析参照例程。

VER=1 OccupancyGridMap\chMapName.eai 和 OccupancyGridMap\chMapName.png 。

chMapName.eai 为 32 字节参数。图片文件是.PNG 格式。

10. 获取任务

```
typedef int(__stdcall *GetMissionByName)(char *chMissionName);
{
chMissionName：任务名称。
}
```

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回成功后，读取文件夹所有内容：Mission\chMissionName

chMissionName.eai：任务描述文件。每行对应的数据为（注意顺序）：

{

uint64 timestamp \r\n

String name \r\n

bool loop \r\n

bool auto_charge \r\n

String TargetName1, TargetName2, \r\n 充电点多个

float charge_threshold\r\n 电量低于此时自动回充。

float min_charge_time \r\n 最短充电时间

float min_charge_percent \r\n 充电最小百分比

string map \r\n

}

示例：

1544705483174

1

1

1

C,

60.00

80.00

40.00

office

Target.eai：目标点描述文件。每行对应一个 Target 点，具体的数据为(Pose2D 3 个数据项用#分隔)：

{

uint64 timestamp, string name, Pose2D pose(float x # float y # float phi) \r\n

}

Pose2D {

float x ;

float y ;

float phi ;

}

示例：

0,desktop1,(-7.30#-1.92#0.00)

0,desktop2,(-10.40#-7.23#1.57)

Path.eai：路径描述文件。每行对应一个 Path，Path 由多个 Pose2D 组成，具体的数据为(Pose2D 3 个数据项用#分隔)：

{

uint64 timestamp, string name, Path paths [Pose2D pose(float x # float y # float phi)&Pose2D pose(float x # float y # float phi)] \r\n

}

示例：

0,todesktop1,[(1.00#2.00#3.00)&(2.00#2.30#2.40)&]

VirPoints.eai：虚拟墙描述文件。对应的数据为：

{

uint64 timestamp \r\n

float resolution \r\n

uint32 width \r\n

uint32 height \r\n

Pose2D origin \r\n

uint32 prefer, uint32 prefer,uint32 prefer \r\n

uint32 unprefer ,uint32 unprefer, uint32 unprefer \r\n

uint32 forbid ,uint32 forbid ,uint32 forbid \r\n

}

示例：

0

0.05

1280

1120

-52.20,-28.20,0.00

11,12,13,14,15,16,17,

111,112,113,114,115,116,117,

704923,704924,704925,704926,704927,704928,704929,704930,704931,704932,704933,704934,704935,704936,704937,704938,704939,704940,704941,704942,704943,704944,704945,704946,704947,704948,704949,704950,704951,704952,704953,704954,704955,704956,704957,704958,704959,704960,704961,704962,704963,704964,704965,704966,704967,704968,704969,704970,704971,704972,704973,704974,704975,704976,704977,704978,7049

79,704980,704981,704982,704983,704984,704985,704986,704987,704988,704989,704990,704991,704992,704993,704994,704995,704996,704997,704998,704999,

Action.eai：动作描述文件。每行对应一个 Action，Action 有不同类型和状态，一行的第 3 个字段值由第 1 个字段类型来区别。：

{

ActionType type, ActionState state , (Waiting waiting or GoToTarget gototarget or Moving moving or Tracking tracking or Docking docking or Charging charging) \r\n

}

相关结构体和 enum 如下：

```
enum ActionType {
   WAITING       = 1;
   GOTOTARGET = 2;
   MOVING         =3;
   TRACKING      = 4;
    GOCHARGE     = 5;
   CHARGING      = 6;
}
```

```
enum ActionState {
    TODO    = 0;
    DOING = 1;
    DONE    = 2;
}

Waiting {
    float duration ;
}
GoToTarget {
    string name ;
}
Moving {
    float angle       ;
    float distance ;
    float max_v       ;
    float max_w       ;
}
Tracking {
    string name ;
    Bool bypass_obstacle ;/是否绕过障碍物?选 0 是停止!
```

}

Charging {

float duration ; // minutes.

float percent ;

}

示例：

1,0,(desktop1)

0,0,(5.00)

3,0,(todesktop1)

2,0,(1.57,2.50,0.50,0.02)

5,0,(30.50,80.50)

11. 状态控制

```
typedef int(__stdcall *StatusControl)(int iStatus, char *pName);
```

{

iStatus：0，空闲 Idle；1，建图 Mapping；2，创建任务 CreateMission；3，导航到目标点 GoToTarget；4， 执行任务 ExecuteMission ；5，开始执行任务 Execute。

pName：创建地图的名称/创建任务的名称（Missioname:MapName）/目标点的名称/执行任务的名称。7,扩展建图，地图名为原图名。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

```
typedef enum _StatType
{
    Idle            = 0,
    Mapping         = 1,
    CreateMission   = 2,
    GoToTarget      = 3,
    ExecuteMission  = 4,
    Execute         = 5,
    Exception       = 6,
    Extend          = 7
}StatType;
```

12. 保存地图

```
typedef int(__stdcall *SaveMapByName)(int iMode,char *chMapName);
```

iMode：1，保存 Server 内存中地图（建图完成）；0，保存 Client 修改后的地图（OccupancyGridMap\chMapName.eai）

{

chMapName：地图名称。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

```
typedef int(__stdcall *SaveMapByNameAndData)(int iMode, char *chMapName, char *pData, UINT32 u32DataLen);
```

{

iMode：1，保存 Server 内存中地图（建图完成）；0，保存 Client 修改后的地图（OccupancyGridMap\chMapName.eai）

chMapName：地图名称。

pData: 读取地图文件中到 BUFF 中的指针。

u32DataLen：buff 数据长度（文件长度）。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

注：如果 pData=NULL，u32DataLen=0，DLL 将读取文件内容。

13. 获取状态

```
typedef int(__stdcall *GetStatus)(void);
```

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

在 PUB 中消息中接收状态：回调或自定义消息

```
typedef struct
{
    UINT64                  u64Timestamp;
    char                    chMissionName[LEN_NAME_MISSION];
    int                     iStat;
    UINT32                  u32Action_id;
    int                     iAction_type;
    int                     iAction_state;
}tsDATA_Stat_Info;
```

14. 删除资源

```
typedef int(__stdcall *DeleteSourceByName)(int iMode, char *pName);
{
```

iMode：1,删除地图； 2，删除任务； 3，删除目标点； 4，删除路径； 5，删除虚拟墙。6，删除数据包，(VER=1)删除手画路径

pName：资源名称。当 iMode 为 3 时，名称为（任务名:目标点）："mission_name:target_name"；当 iMode 为 4 时，名称为（任务名:路径）："mission_name:path_name"；当 iMode 为 5 时，名称为任务名称。

VER=1 时 chName：资源名称即可 无需要 加 任务名：

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

15. 保存目标点

```
typedef int(__stdcall *SaveTargetByName)(char *chMapName, char *chMissionName,
char *chTargetName,char *chData);
```

{

chMapName：地图名称

chMissionName：任务名称。

chTargetName：目标点名称。

chData：文本文件中 Target 格式数据

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

注：在调用成功之后，需要将 Target 保存到对应的任务目录下的 Mission\chMissionName\Target.eai 文件，也可以获取任务 API 详情。且名称在同一个任务下唯一。

示例：Mission\transport\Target.eai

16. 保存路径

```
typedef int(__stdcall *SavePathByName)(char *chMapName, char *chMissionName,
char *chPathName, char *chData);
{
```

chMapName：地图名称

chMissionName：任务名称。

chPathName：路径名称。

chData：文本文件中 Path 格式数据

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

注：在调用成功之后，需要将 Path 保存到对应的任务目录下的 Mission\chMissionName\Path.eai 文件，也可以获取任务 API 详情。且名称在同一个任务下唯一。

示例：Mission\transport\Path.eai

17. 保存虚拟墙

```
typedef int(__stdcall *SaveVirPointsByName)(char *chMapName, char *chMissionName, char *chData);
```

{

chMapName：地图名称

chMissionName：任务名称。

chData：文本文件中 VirPoints 格式数据

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

注：在调用成功之后，需要将 VirPoints 保存到对应的任务目录下的 Mission\chMissionName\VirPoints.eai 文件，也可以获取任务 API 详情。

示例：Mission\transport\VirPoints.eai

18. 保存任务

```
typedef int(__stdcall *SaveMissionByName)(char *chMapName, char *chMissionName, char *chParaData, char *chActionData);
```

{

chMapName：地图名称

chMissionName：任务名称。

chParaData：文本文件 chMissionName.eai 中任务格式数据

chActionData：文本文件 Action.eai 中任务格式数据

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

注：在调用成功之后，需要将 VirPoints 保存到对应的任务目录下的 Mission\chMissionName\chMissionName.eai 文件 Mission\chMissionName\Action.eai 和，也可以获取任务 API 详情。

19. 发送初始位置

```
typedef int(__stdcall *SetBotPose)(float fX, float fY, float fPhi);
```

{

fX：坐标 X

fY：坐标 Y

fPhi：为 0。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

20. 方向控制

```
typedef int(__stdcall *DirectionControl)(float fLinear_x, float fAngle_z);
```

{

fLinear_x：行走线速度，范围：-0.9 - 0.9 。正值向前行走，负值后退。

fAngle_z：旋转角速度，范围：-0.6 - 0.6。正值向右旋转，负值向左旋转。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

21. 获取系统版本号

typedef int(__stdcall *GetSystemVersion)(char *chData);

{

chData：用于接收版本信息指针，请分配最大 1K 的长度。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

22. 初始化机器人（VER=1 使用）

typedef int(__stdcall *SetBotPoseB)(int iX, int iY, float fPhi);

{

iX：栅格化坐标。

iY：栅格化坐标。

FPhi:角度

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

23. 初始化机器人（VER=1 使用）

typedef int(__stdcall *SetBotInitPoseB)(char* chTargetName);

{

chTargetName: 目标点名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

24. 初始化状态（VER=1 使用）

typedef int(__stdcall *GetBotInitPoseStatusB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；5，未完成

}

25. 选择地图（VER=1 使用）

typedef int(__stdcall *SelectMapByName)(char* chMapName);

{

chMapName: 地图名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

26. 保存目标点（VER=1 使用）

typedef int(__stdcall *SaveTargetByNameB)(char* chName, int iType);

{

chName: 目标点名称。机器人当前所在位置

iType：类型。0 初始点，1 充电点，2 导航点

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

27. 获取目标点列表（VER=1 使用）

typedef int(__stdcall *GetTargetsB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

返回结果，存在文件 OccupancyGridMap\chMapName\Target.eai 文件中

多行，每行代表一个点 点的类型分：:0 初始点，1 充电点，2 导航点。

每行格式：时间戳,名称,(栅格化 x#栅格化 y#角度),类型;

如：1543229468,Origin,(256#95#2.3722415552169633),0

28. 增加目标点（VER=1 使用，类型为 2）

typedef int(__stdcall *AddTargetB)(char* chName, int iX, int iY, float fPhi);

{

chName: 目标点名称。

iX：栅格化坐标 X

iY:栅格化坐标 Y

FPhi:角度

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

29. 导航到目标点（VER=1 使用）

typedef int(__stdcall *NavigateTargetB)(char* chTargetName);

{

chTargetName：目标点名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

30. 导航到任意点（VER=1 使用）

typedef int(__stdcall *NavigateTargetAnyB)(int iX, int iY, float fPhi);

{

iX：栅格化坐标 X

iY:栅格化坐标 Y

FPhi:角度

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

31. 导航控制（VER=1 使用）

typedef int(__stdcall *NavigateTargetControlB)(int iMode);

{

iMode：0 暂停 ；1 恢复；2 停止。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

32. 动作列表（VER=1 使用）

typedef int(__stdcall *PathAcitionListB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

成功后，读取文件 SourceList\Action.eai 动作类型如下。动作名称,[(子动作名称#类型)&]。action 的列表数据跟添加手画路径里的 action 数据格式一样，只是 value 值是空的，只需要把值填进去，拼凑到手画路径数据即可

ModifyConfig,[(namespace#string)&(value_b#bool)&(value_i32#int32)&(value_d#double)&(value_s#string)&]

Pause,[(millisecond#uint64)&]

PlaySound,[(sound#string)&(language#string)&(loop#bool)&(stop_all#bool)&]

Rotate2d,[(rotation#double)&]

RotateTo2d,[(rotation#double)&]

StopSound,[(sound#string)&(language#string)&]

Wander,[(distance#double)&]

33. 验证两点是否可以生成线段（VER=1 使用）

typedef int(__stdcall *PathVerifyTwoPointGenLineB)(int iX1, int iY1, int iX2, int iY2, float

fRadius);

{

iX1：栅格化坐标 1 X

iY1: 栅格化坐标 1 Y

iX2：栅格化坐标 2　X

iY2: 栅格化坐标 2　Y

fRadius:曲率

}

API　Indication：

{

Return：0，成功；　1，失败；　2，网络中断；3，未登录；；5 不可以

}

34. 验证多条线是否可以生成路径（VER=1 使用）

typedef int(__stdcall *PathVerifyMultiLinesGenPathB)(char* chMultiLines);

{

chMultiLines：数据格式：Line [(起点坐标)&(终点坐标)&曲率半径]; 例：

[(120#120)&(130#130)&0],[(130#130)&(233#331)&0]。

}

API　Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

35. 生成手画路径（VER=1 使用）

typedef int(__stdcall *PathGenGraphPathB)(char* chName, char* chData);

{

chName：路径名称

chData： 数据格式：分 4 行

// 第 1 行是 points,即点，每个点的数据有名字，栅格化坐标，和动作列表即到达此点时要执行的动作，每个动作有名字和成员，成员是嵌套类型即成员嵌套成员， 成员的数据有名字，类型，和值，动作目前有 Pause 暂停, Rotate2d 平面旋转, RotateTo2d 旋转到角度,PlaySound 播放音乐, StopSound 停止播放, OperateDevice 操作设备。

//第 2 行是 lines，即线，每条线包含名字，起点，终点，和曲率半径。

//第 3 行是 paths，即路径，每条路径包含名字，线段名字列表，点列表，此处点列表不仅只有名字，是全部点的数据，为了不同路径相同点可以执行不一样的动作。

//第 4 行是 pathGroups，即路径组，每个路径组包含名字和路径名字列表

// Line 1 points :

[[[[(name#type#value)&]*actionname]@]$(x#y)$pointname],[[[[(name#type#value)&]*actionname]@]$(x#y)$pointname]

//Line 2 lines : (begin#end#name#radius),(begin#end#name#radius)

//Line 3 paths :

{(linename#linename)?pathsname?points}!{(linename#linename)?pathsname?points}

//Line 4 pathGroups:

[pathsname&(pathname#pathname)&pathstype],[pathsname&(pathname#pathname)&pathstype]

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；5，路径不安全

}

36. 更新手画路径（VER=1 使用）

typedef int(__stdcall *PathUpdateGraphPathB)(char* chName, char* chData);

{

chName：路径名称

chData： 数据格式：分 4 行 同生成手画路径

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

37. 验证手画路径（VER=1 使用）

typedef int(__stdcall *PathVerifyGraphPathB)(char* chName, char* chData);

{

chName：路径名称

chData： 数据格式：分 4 行 同生成手画路径

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

38. 获取手画路径列表（VER=1 使用）

typedef int(__stdcall *PathGetGraphPathB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

获取成功后，数据存在文件 OccupancyGridMap\chMapName\GraphPath\ 文件夹中，每个.EAI 文件就是一个手画路径。

39. 删除手画路径（VER=1 使用）

typedef int(__stdcall *PathDeleteGraphPathB)(char* chName);

{

chName：手画路径名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

40. 重命名手画路径（VER=1 使用）

typedef int(__stdcall *PathRenameGraphPathB)(char* chName, char* chNewName);

{

chName：手画路径名称

chNewName：新名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

41. 获取路径列表（VER=1 使用）

typedef int(__stdcall *PathGetPathB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

获取成功后，数据存在文件 OccupancyGridMap\chMapName\pathlist.eai 文件中。每行一条路径：名称,时间

main_path,2018-11-26 17:57:15

42. 删除路径（VER=1 使用）

typedef int(__stdcall *PathDeletePathB)(char* chName);

{

chName：路径名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

43. 重命名路径（VER=1 使用）

typedef int(__stdcall *PathRenamePathB)(char* chName, char* chNewName);

{

chName：路径名称

chNewName：新名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

44. 开始录制路径（VER=1 使用）

typedef int(__stdcall *PathStartRecordPathB)(char* chName, int iMode);

{

chName：路径名称

iMode：0 为路径，1 为区域路径

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

45. 控制录制路径（VER=1 使用）

typedef int(__stdcall *PathControlRecordPathB)(int iMode, int iStatus);

{

iMode：0 为路径，1 为区域路径

iStatus： 0 停止并保存路径，1 取消录制路径

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

46. 添加动作中（导航点）（VER=1 使用）

typedef int(__stdcall *PathAddPathActionPointB)(char* chName);

{

chData：动作数据。格式： [[(name#type#value)&]*name],

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

47. 获取路径详情（VER=1 使用）

typedef int(__stdcall *PathGetPathInfoB)(char* chName);

{

chName：路径名称

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

得到的是一个路径数组，单条路径则数组里只有一个，区域路径数组中含多个. 保存到 OccupancyGridMap\chMapName\path\chName.eai 文件中

name:[[[[(fields#name#type#value)&(fields#name#type#value)]*actionname]@]$angle$(x#y)$index$pointname],

48. 获取虚拟墙（VER=1 使用）

typedef int(__stdcall *GetVirPointInfoB)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

获取虚拟墙.数据有圆形，线段，封闭多边形，不封闭多边形，矩形; 圆形即原点和半径，线段即起点的终点，矩形即对角点，多边形即多个点. 保存到 OccupancyGridMap\chMapName\VirPoints.eai 文件中

/////分 5 行

//第 1 行 圆形： [(x#y)&radius],

//第 2 行 线段： [(startX#startY)&(endX#endY)],

//第 3 行 封闭多边形： [(x1#y1)&(x2#y2)&(x3#y3)&(x4#y4)&(x5#y5)&],

//第 4 行 不封闭多边形： [(x1#y1)&(x2#y2)&(x3#y3)&(x4#y4)&(x5#y5)&],

//第 5 行 矩形： [(startX#startY)&(endX#endY)],

49. 更新虚拟墙（VER=1 使用）

typedef int(__stdcall *UpdateVirPointInfoB)(char* chData);

{

chData:所有虚拟墙数据。格式同 getVirPointInfoB 返回的数据。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

50. 添加任务队列（VER=1 使用）

typedef int(__stdcall *AddMissionB)(char* chName, char* chData, int iLoop)；

{

chName:名称。

chData：任务内容；4 种任务：PlayPathTask，PlayGraphPathTask，PlayGraphPathGroupTask，NavigationTask。

//[PlayPathTask#map_name#path_name],[PlayGraphPathTask#map_name#graph_name#graph_path_name],[PlayGraphPathGroupTask#map_name#graph_name#graph_path_group_name],[NavigationTask#map_name#position_name],

iLoop：0 不循环，1 循环。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

51. 任务控制

typedef int(__stdcall *MissionControl)(int iStatus);

{

iStatus：0 停止任务/所有任务队列 ；1 暂停任务/所有任务队列；=2 恢复 任务任务/任务队列 ；当 VER=1 时 =3，停止当前任务

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

52. 执行队列（VER=1 使用）

typedef int(__stdcall *StartMissionB)(char* chName,int iLoop,int iLoopCount);

{

chName:任务队列名称。

iLoop: 0 不循环 1 循环；

iLoopCount: 循环次数

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

53. 关机

typedef int(__stdcall *CloseSystem)(void);

{

}

API　Indication：

{

Return：0，成功；　1，失败；　2，网络中断；3，未登录；

}

在充电状态才可关机，关机会关掉上位机，下位机只保留电源，关掉电机，关掉激光

54. 任务完成状态

typedef int(__stdcall *GetMissionIsFinished)(void);

{

}

API　Indication：

{

Return：0，成功；　1，失败；　2，网络中断；3，未登录；5 未完成

}

55. 电机控制

typedef int(__stdcall *MotorControl)(int iMode);

{

iMode：iMode 1 表示开，0 表示关

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

56. 电机控制

typedef int(__stdcall *SpeedControl)(int iMode, int iLevel);

{

iMode： 0 导航速度，1 跑路径速度

iLevel 三级，低速，中速，高速，分表是 0,1,2

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

57. 获取参数

typedef int(__stdcall *GetParameters)(void);

{

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

参数保存到程序安装目录的 EAIParameter.eai 文件，VER=1 时，文件名为：EAIParameterVer1.eai

58. 保存参数

typedef int(__stdcall *SaveParameters)(char *chData);

{

chData：参数内容

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

参数保存到程序安装目录的 EAIParameter.eai 文件， VER=1 时， 文件名为：EAIParameterVer1.eai

59. 保存参数

typedef int(__stdcall *EAIDownloadFile)(int iType, char *chName, BOOL bReDownload);

{

iType：文件类型。1 .bag

chName：文件名称

bReDownload：是否重新下载。

}

API Indication：

{

Return：0，成功； 1，失败； 2，网络中断；3，未登录；

}

下载文件保存在 SourceList 目录下对应的 chName 子目录下。

关于作者：

李光永 konyun@eaibot.com

手机：14775541001