Multi-in-one Card Reader Device SDK

1.23.327.1

制作者 Doxygen 1.8.18

Chapter 1

弃用列表

成员 **mio_enumerate (char *devs_list)**

Chapter 2

模块索引

2.1 模块

这里列出了所有模块:

Chapter 3

文件索引

3.1 文件列表

这里列出了所有文档化的文件，并附带简要说明:

Chapter 4

模块说明

4.1 Error_Code_Definitions

错误码定义

枚举

- enum _TagErrCode {
  MIO_ERR_SUCCESS = 0, MIO_ERR_COM_OPEN_FAIL = -1, MIO_ERR_COM_CLOSE_FAIL = -2,
  MIO_ERR_COM_TIMEOUT = -3,
  MIO_ERR_COM_NO_ACK = -4, MIO_ERR_COM_XCHG_DATA = -5, MIO_ERR_ARG_TOO_SHORT = -6,
  MIO_ERR_ARG_TOO_LONG = -7,
  MIO_ERR_EEPROM_RW = -8, MIO_ERR_KEY_INDEX = -9, MIO_ERR_KEY_LENGTH = -10, MIO_ERR_INPUT_DATA
  = -11,
  MIO_ERR_KEY_NO_ZMK = -12, MIO_ERR_KEY_NO_ZPK = -13, MIO_ERR_KEY_NO_ZMK_ZPK = -14,
  MIO_ERR_KEY_VERIFY = -15,
  MIO_ERR_COUNT_GRAMA = -16, MIO_ERR_DATA_LEN_GRAMA = -17, MIO_ERR_PROC_CANCEL = -18,
  MIO_ERR_COM_WRONGNUM = -19,
  MIO_ERR_COM_HANDLE = -20, MIO_ERR_ARG_NULL_PTR = -21, MIO_ERR_DATA_CHECK_XOR = -22,
  MIO_ERR_NO_ENOUGH_MEM = -23,
  MIO_ERR_IC_POWEROFF_FAIL = -24, MIO_ERR_IC_CARDPRESENT_FAIL = -25, MIO_ERR_IC_APDU_FAIL
  = -26, MIO_ERR_IC_POWERON_FAIL = -27,
  MIO_ERR_PWD_NOT_GHC = -28, MIO_ERR_IC_DAT_NOT_FIND = -29, MIO_ERR_IC_PSE_DATA = -30,
  MIO_ERR_IC_FIND_AID = -31,
  MIO_ERR_IC_SELECT_AID = -32, MIO_ERR_IC_GPO = -33, MIO_ERR_IC_NO_ALLINFO_REC = -34,
  MIO_ERR_IC_NO_ALLINFO_GETDATA = -35,
  MIO_ERR_IC_NO_THIS_TAG = -36, MIO_ERR_IC_APDU_UNSUPPORT = -37, MIO_ERR_ARQC_DATAFIELD
  = -38, MIO_ERR_ARQC_IN_TRANSDATA = -39,
  MIO_ERR_ARQC_FAIL = -40, MIO_ERR_FIELD55_DATA = -41, MIO_ERR_TRANSDETAIL_FORMAT = -42,
  MIO_ERR_TRANSDETAIL_REC = -43,
  MIO_ERR_TRANSDETAIL_REC_DATA = -44, MIO_ERR_ARPC_NOT_FOUND = -45, MIO_ERR_SCRPTTEMP_NOT_FOUND
  = -46, MIO_ERR_SCRPT_NOT_FOUND = -47,
  MIO_ERR_SCRPTMARK_NOT_FOUND = -48, MIO_ERR_EXTAUTH_FAIL = -49, MIO_ERR_SCRPT_EXEC_FAIL
  = -50, MIO_ERR_DATA_FORMAT_REC = -51,
  MIO_ERR_BUF_NOT_ENOUGH = -52, MIO_ERR_LOADDETAIL_FORMAT = -53, MIO_ERR_LOADDETAIL_REC
  = -54, MIO_ERR_LOADDETAIL_REC_DATA = -55,
  MIO_ERR_READ_VER_FAIL = -56, MIO_ERR_9000_FAIL = -57, MIO_ERR_STATE_FAIL = -58, MIO_ERR_EXEC_FAIL
  = -59,
  MIO_ERR_PHOTODIR = -101, MIO_ERR_WRITE_WLT = -102, MIO_ERR_MAKEPHOTO = -103,
  MIO_ERR_LOADPHOTO = -104,
  MIO_ERR_LOADLIB = -106, MIO_ERR_LOADFUNC = -107, MIO_ERR_FINDID = -111, MIO_ERR_MSG_DATA_NULL
  = -200,
  MIO_ERR_MSG_DATA_TOOLONG = -201, MIO_ERR_MSG_WRITEFAIL = -202 }

4.1.1 详细描述

错误码定义

4.1.2 枚举类型说明

4.1.2.1 _TagErrCode

enum _TagErrCode

#include <error_code.h>

全局错误码定义

枚举值

MIO_ERR_SUCCESS	执行成功
MIO_ERR_COM_OPEN_FAIL	打开串口失败
MIO_ERR_COM_CLOSE_FAIL	关闭串口失败
MIO_ERR_COM_TIMEOUT	设备应答超时
MIO_ERR_COM_NO_ACK	设备无应答
MIO_ERR_COM_XCHG_DATA	数据传输错误
MIO_ERR_ARG_TOO_SHORT	消息长度过短
MIO_ERR_ARG_TOO_LONG	消息长度过长
MIO_ERR_EEPROM_RW	EEPROM读写错
MIO_ERR_KEY_INDEX	密钥组号错
MIO_ERR_KEY_LENGTH	密钥长度错
MIO_ERR_INPUT_DATA	输入数据不合法
MIO_ERR_KEY_NO_ZMK	主密钥ZMK不存在
MIO_ERR_KEY_NO_ZPK	工作密钥ZPK不存在
MIO_ERR_KEY_NO_ZMK_ZPK	密钥不存在（ZMK、ZPK、ZAK）
MIO_ERR_KEY_VERIFY	密钥校验值不一致
MIO_ERR_COUNT_GRAMA	账号域语法检查错
MIO_ERR_DATA_LEN_GRAMA	数据长度域语法检查错
MIO_ERR_PROC_CANCEL	取消操作
MIO_ERR_COM_WRONGNUM	错误的串口号
MIO_ERR_COM_HANDLE	串口未打开
MIO_ERR_ARG_NULL_PTR	参数错误，空指针
MIO_ERR_DATA_CHECK_XOR	数据错误，校验和错误
MIO_ERR_NO_ENOUGH_MEM	内存不足，申请内存失败
MIO_ERR_IC_POWEROFF_FAIL	下电失败
MIO_ERR_IC_CARDPRESENT_FAIL	卡在线失败
MIO_ERR_IC_APDU_FAIL	APDU失败
MIO_ERR_IC_POWERON_FAIL	上电失败
MIO_ERR_PWD_NOT_GHC	不是加密键盘

枚举值

MIO_ERR_IC_DAT_NOT_FIND	未找到所需数据
MIO_ERR_IC_PSE_DATA	读取PSE数据错误
MIO_ERR_IC_FIND_AID	未找到有效的AID
MIO_ERR_IC_SELECT_AID	选择AID失败
MIO_ERR_IC_GPO	GPO失败
MIO_ERR_IC_NO_ALLINFO_REC	在读记录过程中没有准备好所有需要的应该存在的数据
MIO_ERR_IC_NO_ALLINFO_GETDATA	取数据中没有准备好所需要的数据
MIO_ERR_IC_NO_THIS_TAG	无此标签
MIO_ERR_IC_APDU_UNSUPPORT	不支持该APDU命令
MIO_ERR_ARQC_DATAFIELD	ARQC数据域不完整
MIO_ERR_ARQC_IN_TRANSDATA	ARQC传入交易数据为空
MIO_ERR_ARQC_FAIL	ARQC失败
MIO_ERR_FIELD55_DATA	55数据域不完整
MIO_ERR_TRANSDETAIL_FORMAT	读取交易明细格式失败
MIO_ERR_TRANSDETAIL_REC	读取交易明细记录失败
MIO_ERR_TRANSDETAIL_REC_DATA	交易明细记录数据长度错误
MIO_ERR_ARPC_NOT_FOUND	没找到ARPC
MIO_ERR_SCRPTTEMP_NOT_FOUND	没找到脚本模板
MIO_ERR_SCRPT_NOT_FOUND	没找到脚本
MIO_ERR_SCRPTMARK_NOT_FOUND	没找到发卡行脚本标识
MIO_ERR_EXTAUTH_FAIL	外部认证失败
MIO_ERR_SCRPT_EXEC_FAIL	脚本执行失败
MIO_ERR_DATA_FORMAT_REC	记录数据格式错误
MIO_ERR_BUF_NOT_ENOUGH	数据缓冲区不足
MIO_ERR_LOADDETAIL_FORMAT	读取圈存明细格式失败
MIO_ERR_LOADDETAIL_REC	读取圈存明细记录失败
MIO_ERR_LOADDETAIL_REC_DATA	圈存明细记录数据长度错误
MIO_ERR_READ_VER_FAIL	读固件版本失败
MIO_ERR_9000_FAIL	判断9000失败
MIO_ERR_STATE_FAIL	状态码错误
MIO_ERR_EXEC_FAIL	执行失败
MIO_ERR_PHOTODIR	传入的生成头像路径不存在
MIO_ERR_WRITE_WLT	写tmp.wlt失败
MIO_ERR_MAKEPHOTO	生成头像失败
MIO_ERR_LOADPHOTO	加载头像失败
MIO_ERR_LOADLIB	加载二代证依赖库失败
MIO_ERR_LOADFUNC	加载二代证依赖库函数失败
MIO_ERR_FINDID	未找到身份证
MIO_ERR_MSG_DATA_NULL	写入磁卡数据为空
MIO_ERR_MSG_DATA_TOOLONG	写入数据过长
MIO_ERR_MSG_WRITEFAIL	写卡失败

4.2 Multi-in-one_Card_Reader_Library

多功能读卡器接口函数库

模块

- Error_Code_Definitions
  错误码定义
- Macro_Definitions
  宏定义
- Version_Information_and_Status_Indicator
  版本信息与状态指示
- Device_Enumeration_and_Open_Close_Operation
  设备枚举与打开关闭操作
- Mag-Stripe_Operation
  磁卡操作接口
- CICC_and_PICC_CPU_Card_Operation
  接触式与非接触式*CPU*卡操作接口
- ID_Card_Operation
  二代身份证操作接口
- RF125KHz_EMID_Card_Operation
  *125KHz*射频卡操作接口
- ISO14443A_Mifare_Card_Operation
  *Mifare TypeA*射频卡操作接口

宏定义

- #define GHC_MIO_API
  声明*Windows*下的函数调用约定.

4.2.1 详细描述

多功能读卡器接口函数库

4.3 Macro_Definitions

宏定义

宏定义

- #define IDCR_READ_MODE_COMPATIBLE 0x0800u

  二代证读卡（*USB*）采用兼容模式，以支持速度较慢的*PC*或云桌面*(*除非实际测试时，频繁遇到*USB*读身份证信息失败的情况，不建议采用此方式，因会大幅降低读卡速度*)*.
- #define IDCR_READ_MODE_EXT 0x0020u

  二代证读卡，读取附加信息.
- #define IDCR_READ_MODE_REREAD 0x0010u

  二代证读卡，强制重新读取.
- #define IDCR_READ_MODE_COMPOSITE 0x0008u

  二代证读卡，合成正反面照片.
- #define IDCR_READ_MODE_PHOTO 0x0004u

  二代证读卡，读取头像.
- #define IDCR_READ_MODE_TEXT 0x0002u

  二代证读卡，读取文本.
- #define IDCR_READ_MODE_FINGER 0x0001u

  二代证读卡，读取指纹信息.

4.3.1 详细描述

宏定义

4.4 Version_Information_and_Status_Indicator

版本信息与状态指示

函数

- void GHC_MIO_API mio_lib_version (char *version_info)
  获取动态库版本信息（不操作硬件）
- void GHC_MIO_API mio_get_log_path (char *path_log)
  获取日志路径
- int32_t GHC_MIO_API mio_firm_version (size_t handle, uint8_t *pbsVerData, int32_t *piVerLen)
  读取读卡器固件版本
- int32_t GHC_MIO_API mio_beep (size_t handle, uint8_t duration, uint8_t interval, uint8_t beeptimes)
  蜂鸣控制

4.4.1 详细描述

版本信息与状态指示

4.4.2 函数说明

4.4.2.1 mio_beep()

```
int32_t GHC_MIO_API mio_beep (
            size_t handle,
            uint8_t duration,
            uint8_t interval,
            uint8_t beeptimes )
```

```
#include <ghcmio_api.h>
```

蜂鸣控制

参数

in	*handle*	设备句柄;
in	*duration*	鸣叫时长，单位（50ms）;
in	*interval*	间隔时长，单位（50ms）;
in	*beeptimes*	鸣叫次数;

返回

返回值

0	成功
其他	失败

4.4.2.2 mio_firm_version()

```
int32_t GHC_MIO_API mio_firm_version (
             size_t handle,
             uint8_t * pbsVerData,
             int32_t * piVerLen )
```

```
#include <ghcmio_api.h>
```

读取读卡器固件版本

参数

in	*handle*	设备句柄；
out	*pbsVerData*	固件版本信息。

内容包括：

标识	长度	说明
CUP设备信息	8字节	由银联定义的读写器规范版本信息
受理方信息	8字节	由受理方定义的版本信息
Len	1字节	厂家自定义信息长度
厂家自定义信息	Len字节	厂家自定义信息

参数

out	*piVerLen*	固件版本信息长度

返回

返回值

0	成功
其他	失败

4.4.2.3 mio_get_log_path()

```
void GHC_MIO_API mio_get_log_path (
            char * path_log )
```

```
#include <ghcmio_api.h>
```

获取日志路径

参数

out	*path_log*	输出日志路径

4.4.2.4 mio_lib_version()

```
void GHC_MIO_API mio_lib_version (
            char * version_info )
```

```
#include <ghcmio_api.h>
```

获取动态库版本信息（不操作硬件）

参数

out	*version_info*	输出动态库的版本信息

返回值

0	

4.5 Device_Enumeration_and_Open_Close_Operation

设备枚举与打开关闭操作

函数

- int32_t GHC_MIO_API mio_enumerate (char *devs_list)
  枚举主机上的串口和可用的*USB*设备
- int32_t GHC_MIO_API mio_enum_devices (int32_t dev_type, char **p_out_list)
  枚举主机上的串口和可用的*USB*设备
- void GHC_MIO_API mio_free_list (const char *p_list)
  释放*mio_enum_devices*生成的设备列表
- size_t GHC_MIO_API mio_open (const char *psport, char *psexport, int32_t baud)
  打开一个设备
- int32_t GHC_MIO_API mio_close (size_t handle, int32_t iPortTypeRfu)
  关闭一个设备
- int32_t GHC_MIO_API mio_set_baud (size_t handle, uint32_t baud, uint8_t save_mode)
  设置波特率
- size_t GHC_MIO_API mio_open_and_set_baud (const char *psport, char *psexport, int32_t origin_baud, int32_t dest_baud)
  打开一个串口设备并设置波特率, 支持*9600, 19200, 38400, 57600, 115200*
- int32_t GHC_MIO_API mio_reset_device (size_t handle, uint8_t bMute)
  复位设备, 已经建立的连接将丢失重置, 需重新打开设备

4.5.1 详细描述

设备枚举与打开关闭操作

4.5.2 函数说明

4.5.2.1 mio_close()

```
int32_t GHC_MIO_API mio_close (
            size_t handle,
            int32_t iPortTypeRfu )
```

```
#include <ghcmio_api.h>
```

关闭一个设备

参数

in	*handle*	设备句柄；
in	*port_type*	设备类型，0-USB，1-串口；

返回

返回值

<0	- 失败，详见错误代码表
0	- 成功

4.5.2.2 mio_enum_devices()

```
int32_t GHC_MIO_API mio_enum_devices (
            int32_t dev_type,
            char ** p_out_list )
```

```
#include <ghcmio_api.h>
```

枚举主机上的串口和可用的USB设备

参数

in	*dev_type*	要枚举的设备类型，1-"USB HID", 2-"串口", 3-"PCSC", 4-"SDT身份证阅读器"；
out	*p_out_list*	枚举到的设备的集合，内存由驱动分配；

串口设备以"COM"开头（Linux或类Unix系统下，以"ttyS"开头或全路径），USB HID设备以"GHC"开头，多个设备用字符'|'分隔，格式为"序列号1|序列号2|...|序列号n|"，例1："COM1|COM2|COM4|"，表示找到了3个串口设备"COM1、COM2、COM4"；例2："GHC815#112233AABBCCDD|GHC815#2233445566778899|"，表示找到了2个USB设备；

1、列表结尾为字符'|';
2、串口设备为主机上找到的串口号，不代表有串口设备连接，可通过打开设备接口探测设备是否存在；
3、设备列表用完后应调用mio_free_list释放。

注意

返回

返回值

<0	- 失败，详见错误代码表；
0	- 没有符合要求的设备；
>0	- 枚举到1个或多个设备；

4.5.2.3 mio_enumerate()

```
int32_t GHC_MIO_API mio_enumerate (
            char * devs_list )
```

```
#include <ghcmio_api.h>
```

枚举主机上的串口和可用的USB设备

弃用

参见

mio_enum_devices()

参数

out	*devs_list*	输出枚举到的设备序列号的字符串集合；

串口设备以“COM”开头（Linux或类Unix系统下，以“ttyS”开头），USB设备以“GHC”开头， 多个设备用字符‘|’分隔，格式为“序列号1|序列号2|...|序列号n|”， 例如：”COM1|COM2|COM4|GHC815#112233AABBCC↩DD|GHC815#2233445566778899|”，表示找到了3个串口设备“COM1、COM2、COM4”和2个USB设备

1、列表结尾为字符‘|’;
2、串口设备为主机上找到的串口号，不代表有串口设备连接，可通过打开设备接口探测设备是否存在。

注意

返回

返回值

<0	- 失败，详见错误代码表；
>0	- 枚举到的设备数量；

4.5.2.4 mio_free_list()

```
void GHC_MIO_API mio_free_list (
            const char * p_list )
```

```
#include <ghcmio_api.h>
```

释放mio_enum_devices生成的设备列表

参数

in	*p_list*	设备列表指针；

返回

返回值

<0	- 失败，详见错误代码表
0	- 成功

4.5.2.5 mio_open()

```
size_t GHC_MIO_API mio_open (
            const char * psport,
            char * psexport,
            int32_t baud )
```

```
#include <ghcmio_api.h>
```

打开一个设备

参数

in	*psport*	设备名称，串口设备以“COM”开头（Linux或类Unix系统下，以“ttyS”开头），USB设备以“GHC”开头，传NULL表示打开默认USB设备；
in	*psexport*	串口扩展盒端口，“A”、“B”、“C”、“D”、“K”分别表示A/B/C/D/K口，无扩展盒传“9”或NULL；
in	*baud*	串口波特率，USB设备忽略此参数。支持9600、19200、38400、57600、115200，默认波特率9600；

返回

返回值

<0	- 失败，详见错误代码表;
>0	- 设备句柄，后续对此设备的操作皆基于此句柄;

4.5.2.6 mio_open_and_set_baud()

```
size_t GHC_MIO_API mio_open_and_set_baud (
            const char * psport,
            char * psexport,
            int32_t origin_baud,
            int32_t dest_baud )
```

```
#include <ghcmio_api.h>
```

打开一个串口设备并设置波特率, 支持9600, 19200, 38400, 57600, 115200

参数

in	*psport*	设备名称，串口设备以“COM”开头（Linux或类Unix系统下，以“ttyS”开头），USB设备以“GHC”开头，传NULL表示打开默认USB设备；
in	*psexport*	串口扩展盒端口，“A”、“B”、“C”、“D”、“K”分别表示A/B/C/D/K口，无扩展盒传“9”或NULL；
in	*origin_baud*	原始波特率,打开串口时的波特率，支持9600、19200、38400、57600、115200，默认波特率9600；
in	*dest_baud*	要设置的目标波特率；

返回

返回值

<0	- 失败，详见错误代码表;
>0	- 设备句柄，后续对此设备的操作皆基于此句柄;

4.5.2.7 mio_reset_device()

```
int32_t GHC_MIO_API mio_reset_device (
            size_t handle,
            uint8_t bMute )
```

```
#include <ghcmio_api.h>
```

复位设备, 已经建立的连接将丢失重置, 需重新打开设备

参数

in	*handle*	设备句柄；
in	*bMute*	是否静音复位, 0-非静音, 1-静音；

返回

返回值

<0	- 失败，详见错误代码表
0	- 成功

4.5.2.8 mio_set_baud()

```
int32_t GHC_MIO_API mio_set_baud (
            size_t handle,
            uint32_t baud,
            uint8_t save_mode )
```

```
#include <ghcmio_api.h>
```

设置波特率

参数

in	*handle*	设备句柄；
in	*baud*	波特率索引，0~7={9600, 19200, 38400, 57600, 115200, 230400, 460800, 961600}；
in	*save_mode*	设备重启后是否恢复默认波特率： 0 - 恢复默认波特率， 1 - 保持当前设定的波特率；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.6 Mag-Stripe_Operation

磁卡操作接口

模块

- Mag-Stripe_Async_Operation
  磁卡异步操作接口
- Mag-Stripe_Sync_Operation
  磁卡阻塞模式接口

4.6.1 详细描述

磁卡操作接口

4.7 Mag-Stripe_Async_Operation

磁卡异步操作接口

类型定义

- typedef void(CALLBACK * cb_on_swipe_magstripe) (int code, char *info)
  读磁卡信息回调函数

函数

- int32_t GHC_MIO_API mio_msr_polling_start (size_t handle, int32_t query_interval, cb_on_swipe_magstripe cb)
  启动一个轮询读磁卡的线程,获取到磁条信息时或异常时将通过回调函数反馈
- void GHC_MIO_API mio_msr_polling_interval (int32_t interval_ms)
  设置轮询间隔时间,立即生效
- void GHC_MIO_API mio_msr_polling_stop (void)
  终止轮询读磁卡的线程
- int32_t GHC_MIO_API mio_msr_qm_start (size_t handle)
  启动读磁卡
- int32_t GHC_MIO_API mio_msr_qm_status (size_t handle)
  获取读磁卡状态
- int32_t GHC_MIO_API mio_msr_qm_get (size_t handle, char *track1, char *track2, char *track3, int32_t *track1len, int32_t *track2len, int32_t *track3len)
  获取读到的磁条数据
- int32_t GHC_MIO_API mio_msr_qm_stop (size_t handle)
  结束读卡
- int32_t GHC_MIO_API mio_msr_qm_status_raw (size_t handle)
  获取读磁卡*raw*数据状态
- int32_t GHC_MIO_API mio_msr_qm_get_raw (size_t handle, int32_t index, char *track, int32_t *tracklen)
  获取读到的磁条数据
- int32_t GHC_MIO_API mio_msr_qm_stop_raw (size_t handle)
  结束*raw*读卡
- int32_t GHC_MIO_API mio_msr_raw2str (char *praw, int32_t raw_len, char *pstr)
  将磁条*raw*数据转为字符串
- int32_t GHC_MIO_API mio_msr_get_crypto_card_sn (char *praw, int32_t raw_len, char *pstr)
  将磁条*raw*数据转为卡号

4.7.1 详细描述

磁卡异步操作接口

4.7.2 类型定义说明

4.7.2.1 cb_on_swipe_magstripe

```
typedef void(CALLBACK * cb_on_swipe_magstripe) (int code, char *info)
```

```
#include <ghcmio_api.h>
```

读磁卡信息回调函数

参数

in	*code*	状态码,1-读卡成功,9-检测到刷卡但卡片无有效信息
in	*info*	磁条信息,状态码为1时传入,格式为|分隔的三个磁 道信息,TRACK1|TRACK2|TRACK3,无数据的磁道

返回

void

4.7.3 函数说明

4.7.3.1 mio_msr_get_crypto_card_sn()

```
int32_t GHC_MIO_API mio_msr_get_crypto_card_sn (
            char * praw,
            int32_t raw_len,
            char * pstr )
```

```
#include <ghcmio_api.h>
```

将磁条raw数据转为卡号

参数

in	*praw*	磁条raw数据
in	*raw_len*	磁条raw数据长度，长度为88或176（176是双份数据）；
out	*pstr*	转换后的卡号数据，可能的最大长度为40位字符串。

返回

返回值

0	成功
其他	失败

4.7.3.2 mio_msr_polling_interval()

```
void GHC_MIO_API mio_msr_polling_interval (
            int32_t interval_ms )
```

```
#include <ghcmio_api.h>
```

设置轮询间隔时间,立即生效

参数

in	*interval_ms*	查询间隔时间,单位毫秒,例如:200ms.此参数在一定程 度上关系到检测刷卡的灵敏度,但过小的间隔时间可能会使其 它分时处理任务优先级过

返回

void

4.7.3.3 mio_msr_polling_start()

```
int32_t GHC_MIO_API mio_msr_polling_start (
            size_t handle,
            int32_t query_interval,
            cb_on_swipe_magstripe cb )
```

```
#include <ghcmio_api.h>
```

启动一个轮询读磁卡的线程,获取到磁条信息时或异常时将通过回调函数反馈

参数

in	*handle*	设备句柄
in	*query_interval*	查询间隔时间,单位毫秒,例如:200ms.此参数在一定程 度上关系到检测刷卡的灵敏度,但过小的间隔时间可能会使其 它分时处理任务优先级过
in	*cb*	回调函数

返回

返回值

0	成功
其他	启动失败

4.7.3.4 mio_msr_polling_stop()

```
void GHC_MIO_API mio_msr_polling_stop (
            void  )
```

#include <ghcmio_api.h>

终止轮询读磁卡的线程

返回

void

4.7.3.5 mio_msr_qm_get()

```
int32_t GHC_MIO_API mio_msr_qm_get (
            size_t handle,
            char * track1,
            char * track2,
            char * track3,
            int32_t * track1len,
            int32_t * track2len,
            int32_t * track3len )
```

#include <ghcmio_api.h>

获取读到的磁条数据

参数

in	*handle*	设备句柄
out	*track1*	一磁道数据；
out	*track2*	二磁道数据；
out	*track3*	三磁道数据；
out	*track1len*	一磁道数据长度；
out	*track2len*	二磁道数据长度；
out	*track3len*	三磁道数据长度；

返回

返回值

0	成功
其他	失败

4.7.3.6 mio_msr_qm_get_raw()

```
int32_t GHC_MIO_API mio_msr_qm_get_raw (
```

```
            size_t handle,
            int32_t index,
            char * track,
            int32_t * tracklen )
```

```
#include <ghcmio_api.h>
```

获取读到的磁条数据

参数

in	*handle*	设备句柄
in	*index*	磁道索引，1、2、3；
out	*track*	磁道RAW数据；
out	*tracklen*	磁道数据长度；

返回

返回值

0	成功
其他	失败

4.7.3.7 mio_msr_qm_start()

```
int32_t GHC_MIO_API mio_msr_qm_start (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

启动读磁卡

参数

in	*handle*	设备句柄

返回

返回值

0	成功
其他	启动失败

4.7.3.8 mio_msr_qm_status()

```
int32_t GHC_MIO_API mio_msr_qm_status (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

获取读磁卡状态

参数

in	*handle*	设备句柄

返回

返回值

0	未检测到刷卡
1	检测到刷卡动作
其他	通讯错误

4.7.3.9 mio_msr_qm_status_raw()

```
int32_t GHC_MIO_API mio_msr_qm_status_raw (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

获取读磁卡raw数据状态

参数

in	*handle*	设备句柄

返回

返回值

0	未检测到刷卡
>0	检测到刷卡动作
其他	通讯错误

4.7.3.10 mio_msr_qm_stop()

```
int32_t GHC_MIO_API mio_msr_qm_stop (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

结束读卡

参数

in	*handle*	设备句柄

返回

返回值

0	成功
其他	失败

4.7.3.11 mio_msr_qm_stop_raw()

```
int32_t GHC_MIO_API mio_msr_qm_stop_raw (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

结束raw读卡

参数

in	*handle*	设备句柄

返回

返回值

0	成功
其他	失败

4.7.3.12 mio_msr_raw2str()

```
int32_t GHC_MIO_API mio_msr_raw2str (
            char * praw,
            int32_t raw_len,
            char * pstr )
```

```
#include <ghcmio_api.h>
```

将磁条raw数据转为字符串

参数

in	*praw*	磁条raw数据
in	*raw_len*	磁条raw数据长度，长度为88或176（176是双份数据）；
out	*pstr*	转换后的字符串数据，可能的最大长度为raw_len的 8倍，如果是双份数据中间用'|'分隔。

返回

返回值

0	成功
其他	失败

4.8 Mag-Stripe_Sync_Operation

磁卡阻塞模式接口

函数

- int32_t GHC_MIO_API mio_msr_cancel (size_t handle)

  取消读写磁条
- int32_t GHC_MIO_API mio_msr_erase (size_t handle, int32_t trackselect, int32_t timeout_ms)

  擦除磁条信息
- int32_t GHC_MIO_API mio_msr_read (size_t handle, int32_t trackselect, char *track1, char *track2, char *track3, int32_t timeout)

  读取磁条信息【阻塞方式】
- int32_t GHC_MIO_API mio_msr_write (size_t handle, int32_t trackselect, int32_t delimiter, const char *track1, const char *track2, const char *track3, int32_t timeout_ms)

  写磁条信息

4.8.1 详细描述

磁卡阻塞模式接口

4.8.2 函数说明

4.8.2.1 mio_msr_cancel()

```
int32_t GHC_MIO_API mio_msr_cancel (
            size_t handle )
```

```
#include <ghcmio_api.h>
```

取消读写磁条

参数

in	*handle*	设备句柄

返回

返回值

0	成功
其它	失败

4.8.2.2 mio_msr_erase()

```
int32_t GHC_MIO_API mio_msr_erase (
            size_t handle,
            int32_t trackselect,
            int32_t timeout_ms )
```

```
#include <ghcmio_api.h>
```

擦除磁条信息

参数

in	*handle*	设备句柄
in	*trackselect*	磁道选择: 1:一磁道; 2:二磁道; 3:三磁道; 组合后，23:二三磁道，依此，12、13、123等组合。
in	*timeout*	超时时间【毫秒】

返回

返回值

0	成功
其它	执行失败，返回错误码

4.8.2.3 mio_msr_read()

```
int32_t GHC_MIO_API mio_msr_read (
            size_t handle,
            int32_t trackselect,
            char * track1,
            char * track2,
            char * track3,
            int32_t timeout )
```

```
#include <ghcmio_api.h>
```

读取磁条信息【阻塞方式】

参数

in	*handle*	设备句柄
in	*trackselect*	磁道选择，1：一磁道，2：二磁道； 3：三磁道； 23：二三磁道。

参数

out	*track1*	一磁道数据；
out	*track2*	二磁道数据；
out	*track3*	三磁道数据；
in	*timeout*	超时时间【毫秒】

返回

返回值

0	成功
其它	读取磁条信息失败，返回错误码

4.8.2.4 mio_msr_write()

```
int32_t GHC_MIO_API mio_msr_write (
            size_t handle,
            int32_t trackselect,
            int32_t delimiter,
            const char * track1,
            const char * track2,
            const char * track3,
            int32_t timeout_ms )
```

```
#include <ghcmio_api.h>
```

写磁条信息

参数

in	*handle*	设备句柄
in	*trackselect*	磁道选择: 1:一磁道; 2:二磁道; 3:三磁道; 组合后，23:二三磁道，依此，12、13、123等组合。
in	*delimiter*	保留，传0;
out	*track1*	一磁道数据；
out	*track2*	二磁道数据；
out	*track3*	三磁道数据；
in	*timeout*	超时时间【毫秒】

返回

返回值

0	成功
其它	写磁条信息失败，返回错误码

4.9 CICC_and_PICC_CPU_Card_Operation

接触式与非接触式CPU卡操作接口

模块

- CPU_Card_Basic_Operation
  *CPU*卡基本操作
- PBOC_CPU_Card_Operation
  *PBOC*金融*IC*卡接口

4.9.1 详细描述

接触式与非接触式CPU卡操作接口

4.10 CPU_Card_Basic_Operation

CPU卡基本操作

函数

- int32_t GHC_MIO_API mio_icc_poweron (size_t handle, uint8_t card_type, uint16_t timeout, char *atr, int32_t *length)

  给设备上电
- int32_t GHC_MIO_API mio_icc_poweroff (size_t handle, uint8_t card_type, uint16_t timeout)

  设备下电
- int32_t GHC_MIO_API mio_icc_status (size_t handle, uint8_t card_type)

  获取接触卡状态
- int32_t GHC_MIO_API mio_icc_apdu (size_t handle, uint8_t card_type, const char *psapdu, int32_t apdu_length, char *psrpdu, int32_t *rpdu_len)

  发送*APDU*命令
- int32_t GHC_MIO_API **mio_etc_read_number** (size_t handle, uint8_t card_slot, uint16_t timeout, char *pscard_number)

4.10.1 详细描述

CPU卡基本操作

4.10.2 函数说明

4.10.2.1 mio_icc_apdu()

```
int32_t GHC_MIO_API mio_icc_apdu (
            size_t handle,
            uint8_t card_type,
            const char * psapdu,
            int32_t apdu_length,
            char * psrpdu,
            int32_t * rpdu_len )
```

```
#include <ghcmio_api.h>
```

发送APDU命令

参数

in	*handle*	设备句柄
in	*card_type*	卡片类型： 0x00-接触， 0xFF-非接, 2-自动识别接触非接， 0x10～0x13：PSAM1/PSAM2/PSAM3/PSAM4
in	*psapdu*	要发送的APDU命令
in	*apdu_length*	APDU命令的长度
	psrpdu	返回的数据
	rpdu_len	返回数据的长度

返回

返回值

0	成功
其它	发送APDU命令失败，返回错误码

4.10.2.2 mio_icc_poweroff()

```
int32_t GHC_MIO_API mio_icc_poweroff (
            size_t handle,
            uint8_t card_type,
            uint16_t timeout )
```

```
#include <ghcmio_api.h>
```

设备下电

参数

in	*handle*	设备句柄
in	*card_type*	卡片类型: 0x00:接触; 0xFF:非接; 2:自动识别接触非接; 0x10～0x13：PSAM1/PSAM2/PSAM3/PSAM4
in	*timeout*	超时时间 单位毫秒

返回

返回值

0	成功
其它	下电失败，返回错误码

4.10.2.3 mio_icc_poweron()

```
int32_t GHC_MIO_API mio_icc_poweron (
            size_t handle,
            uint8_t card_type,
            uint16_t timeout,
            char * atr,
            int32_t * length )
```

```
#include <ghcmio_api.h>
```

给设备上电

参数

in	*handle*	设备句柄
in	*card_type*	卡片类型: 0x00:接触; 0xFF:非接; 2:自动识别接触非接; 0x10～0x13：PSAM1/PSAM2/PSAM3/PSAM4
in	*timeout*	超时时间 单位毫秒
	atr	上电返回数据
	length	上电返回数据的长度

返回

返回值

0	成功
其它	上电失败，返回错误码

4.10.2.4 mio_icc_status()

```
int32_t GHC_MIO_API mio_icc_status (
            size_t handle,
            uint8_t card_type )
```

```
#include <ghcmio_api.h>
```

获取接触卡状态

参数

in	*handle*	设备句柄
in	*card_type*	卡片类型： 0x00-接触， 0xFF-非接, 2-自动识别接触非接， 0x10～0x13：PSAM1/PSAM2/PSAM3/PSAM4

返回

返回接触卡状态

返回值

1	已上电；
2	未上电；

返回值

5	无卡
其它	获取接触卡状态失败，返回错误码

4.11 PBOC_CPU_Card_Operation

PBOC金融IC卡接口

函数

- int32_t GHC_MIO_API mio_pboc_get_card_info (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, const char *psTagList, uint8_t *pbsCardType, uint8_t *psUserInfo, uint16_t *wInfoLen, uint8_t bXtraProc)
  金融*IC*卡 获取卡片信息
- int32_t GHC_MIO_API **mio_pboc_get_arqc** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, uint16_t wAidLen, const uint8_t *psTxDat, uint16_t wTxDatLen, uint8_t *psArqc, uint16_t *pwArqcLen, uint8_t bIsNeedPwrOn)
- int32_t GHC_MIO_API **mio_pboc_arpc_exec_script** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const uint8_t *psTxDat, uint16_t wTxDatLen, const uint8_t *psArpc, uint16_t wArpcLen, uint8_t *pbsOut↩Buf, uint16_t *pwOutLen)
- int32_t GHC_MIO_API **mio_pboc_get_trans_detail** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, uint8_t num, const char *aid, uint8_t *psDetail, uint16_t *pwDetailLen, uint8_t bIsNeedPwrOn)
- int32_t GHC_MIO_API **mio_pboc_get_card_info_all** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *psTagList, uint8_t *pbsCardType, uint8_t *pbsOutBuf, uint16_t *pwOutBufLen, uint8_t bIsNeedPwrOn)
- int32_t **mio_pboc_is_app_locked** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, uint8_t bIsNeedPwrOn)
- int16_t GHC_MIO_API **mio_pboc_get_load_detail** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, uint8_t num, const char *aid, uint8_t *psDetail, uint16_t *pwDetailLen, uint8_t bIsNeedPwrOn)

4.11.1 详细描述

PBOC金融IC卡接口

4.11.2 函数说明

4.11.2.1 mio_pboc_get_card_info()

```
int32_t GHC_MIO_API mio_pboc_get_card_info (
            size_t handle,
            uint8_t bCardSlot,
            uint16_t wTimeout,
            const char * aid,
            const char * psTagList,
            uint8_t * pbsCardType,
            uint8_t * psUserInfo,
            uint16_t * wInfoLen,
            uint8_t bXtraProc )
```

```
#include <ghcmio_api.h>
```

金融IC卡 获取卡片信息

参数

in	*handle*	设备句柄
in	*bCardSlot*	卡片类型： 0x00:接触， 0xFF:非接, 2:自动识别接触非接，0x10～0x13:PSAM1/PSAM2/PSAM3/PSAM4
in	*wTimeout*	超时时间 单位毫秒
in	*aid*	应用ID，传NULL表示使用默认AID，支持但不限于： A000000333010101-借记，A000000333010102-贷记， A000000333010106-纯电子现金
in	*psTagList*	标签列表,根据自己需要获取的数据传入一组字符组合,例如:传入"AGJ"表示需要获取"应用主账号+电子现金余额+应用主账号序列号"; 注意:传入的标签越多,则读取的数据越多,相对耗时也越多,所以建议仅传入必要的标签; 标签和数据域对应关系如下:

标签	数据域
A	应用主账号
B	姓名
C	证件类型
D	证件号
E	二磁道等效数据
F	一磁道等效数据
G	电子现金余额
H	电子现金余额上限
I	应用失效日期
J	应用主账号序列号
K	应用货币代码
L	应用生效日期
M	电子现金单笔交易限额
N	ATC
O	持卡人姓名扩展
P	终端国家代码

参数

out	*pbsCardType*	激活成功的卡片类型
out	*psUserInfo*	读卡成功返回的卡片数据字段由传入的标签决定, 每个标签的数据格式是:"1位字母标签+3位十进制数据长度+数据(实际数据无+)", 例如:传入的 psTagList 是"ABC",返回数据是"A006123456B003^_^C002FF"
out	*wInfoLen*	返回数据的长度
in	*bXtraProc*	特殊处理，各个bit位定义：

定义	使能位	掩码	**0**	**1**
遍历卡片中可用的应用	bit1	0x02	不遍历	遍历
是否需要对卡片上电	bit0	0x01	不需要	需要

返回

返回值

0	成功
其它	发送APDU命令失败，返回错误码

4.12 ID_Card_Operation

二代身份证操作接口

函数

- int32_t GHC_MIO_API mio_idcr_read_bin (size_t handle, uint32_t read_mode, uint32_t timeout, uint8_t *pabtext, uint32_t *puitextlen, uint8_t *pabwlt, uint32_t *puiwltlen, uint8_t *pabfp, uint32_t *puifplen, uint8_t *pabxtra, uint32_t *puixtralen)

  读取身份证原始数据
- int32_t GHC_MIO_API mio_idcr_read (size_t handle, uint32_t read_mode, uint32_t timeout)

  读身份证并缓存信息
- int32_t GHC_MIO_API mio_idcr_getinfo (size_t handle, uint32_t read_mode, uint32_t timeout, int32_t pictype, char *pstxt, char *psfp, const char *psphotopath, const char *psfrontpath, const char *psbackpath)

  读取身份证信息
- void GHC_MIO_API mio_idcr_set_separation (char separation)

  设置读身份证输出文字信息时的分隔符，默认是'|'分隔，可以设置为其它任意单字节*ASCII*字符
- int32_t GHC_MIO_API mio_idcr_getraw (size_t handle, uint32_t raw_type, char *psbuf, int32_t *pidatlen)

  获取身份证未解码的原始数据
- int32_t GHC_MIO_API mio_idcr_read_samid (size_t handle, uint32_t timeout, char *pssamid)

  读取*SAM*模块*ID*
- int32_t GHC_MIO_API mio_idcr_transive (size_t handle, uint32_t timeout, uint8_t bCmd, uint8_t bArg, uint32_t *pdwRspCode, uint8_t *psRspData, int32_t iRspBufSize, int32_t *piRspDataLen)

  收发*AA*指令
- int32_t GHC_MIO_API mio_idcr_read_uid (size_t handle, uint32_t timeout, char *psuid)

  读取身份证物理*ID*
- int32_t GHC_MIO_API mio_idcr_read_dn (size_t handle, uint32_t timeout, char *psdn)

  读二代证*DN*码
- int32_t GHC_MIO_API mio_idcr_typeb_apdu (size_t handle, uint32_t timeout, uint8_t *pscmd, uint16_t cmd_len, uint8_t *psrsp, uint16_t *p_rsp_len)

  ISO14443 TypeB apdu 交互

4.12.1 详细描述

二代身份证操作接口

4.12.2 函数说明

4.12.2.1 mio_idcr_getinfo()

```
int32_t GHC_MIO_API mio_idcr_getinfo (
            size_t handle,
            uint32_t read_mode,
            uint32_t timeout,
            int32_t pictype,
            char * pstxt,
            char * psfp,
            const char * psphotopath,
            const char * psfrontpath,
            const char * psbackpath )
```

```
#include <ghcmio_api.h>
```

读取身份证信息

参数

in	*handle*	设备句柄；
in	*read_mode*	读取模式，可传入下述多个模式的组合；

读取模式	使能位	掩码	宏定义
兼容模式	bit11	0x0800	IDCR_READ_MODE_COMPATIBLE
附加信息	bit5	0x0020	IDCR_READ_MODE_EXT
强制重新读取	bit4	0x0010	IDCR_READ_MODE_REREAD
正反面	bit3	0x0008	IDCR_READ_MODE_COMPOSITE
头像	bit2	0x0004	IDCR_READ_MODE_PHOTO
文字	bit1	0x0002	IDCR_READ_MODE_TEXT
指纹	bit0	0x0001	IDCR_READ_MODE_FINGER

注意

例如： 需要文字(bit1)、头像(bit2)和正反面信息(bit3)，则传入**14(0x0E)**;
或者传入多个宏的或逻辑组合：
(IDCR_READ_MODE_COMPOSITE | IDCR_READ_MODE_TEXT | IDCR_READ_MODE_PHOTO);

参数

in	*timeout*	等待超时，毫秒；
in	*pictype*	生成图像类型，0-**BMP**，1-**JPG**，2-**PNG**；
out	*pstxt*	文字信息缓冲区；

匹配身份证和港澳台居民居住证：
证件类型代码(身份证A，港澳台J)|
姓名|
性别|
民族|
出生年月日|
住址|
公民身份号码|
签发机关|
有效期限起始日期|
有效期限结束日期|
有效期限|
预留 1|
证件版本号|
预留 2|
预留 3

匹配外国人永久居住证返回格式：
证件类型代码(I)|
英文姓名|
性别|
国籍或地区代号|
出生年月日|
预留|
永久居住证号码|
签发机关|
有效期限起始日期|
有效期限结束日期|
有效期限|
中文姓名|

证件版本号|
预留 1|
预留 2

参数

out	*psfp*	指纹信息缓冲区，长度2048字节；
in	*psphotopath*	头像图像文件名绝对路径；
in	*psfrontpath*	合成证件正面图像文件名绝对路径；
in	*psbackpath*	合成证件反面图像文件名绝对路径；

返回

返回值

0	- 成功;
<0	- 失败，详见错误代码表;

4.12.2.2 mio_idcr_getraw()

```
int32_t GHC_MIO_API mio_idcr_getraw (
            size_t handle,
            uint32_t raw_type,
            char * psbuf,
            int32_t * pidatlen )
```

```
#include <ghcmio_api.h>
```

获取身份证未解码的原始数据

参数

in	*handle*	设备句柄；
in	*raw_type*	取回数据的类别，每次调用取回一种数据， 支持的类型有：

读取模式	使能位	掩码	宏定义
附加信息	bit5	0x0020	IDCR_READ_MODE_EXT
头像	bit2	0x0004	IDCR_READ_MODE_PHOTO
文字	bit1	0x0002	IDCR_READ_MODE_TEXT
指纹	bit0	0x0001	IDCR_READ_MODE_FINGER

参数

out	*psbuf*	输出缓冲区，由调用者分配；
out	*pidatlen*	数据长度；

返回

返回值

0	- 成功;
<0	- 失败，详见错误代码表;

4.12.2.3 mio_idcr_read()

```
int32_t GHC_MIO_API mio_idcr_read (
            size_t handle,
            uint32_t read_mode,
            uint32_t timeout )
```

```
#include <ghcmio_api.h>
```

读身份证并缓存信息

参数

in	*handle*	设备句柄；
in	*read_mode*	读取模式

参见

mio_idcr_getinfo()

参数

in	*timeout*	等待超时，毫秒；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.4 mio_idcr_read_bin()

```
int32_t GHC_MIO_API mio_idcr_read_bin (
            size_t handle,
            uint32_t read_mode,
            uint32_t timeout,
            uint8_t * pabtext,
            uint32_t * puitextlen,
            uint8_t * pabwlt,
            uint32_t * puiwltlen,
            uint8_t * pabfp,
            uint32_t * puifplen,
            uint8_t * pabxtra,
            uint32_t * puixtralen )
```

```
#include <ghcmio_api.h>
```

读取身份证原始数据

参数

in	*handle*	设备句柄；
in	*read_mode*	读取模式

参见

mio_idcr_getinfo()

参数

in	*timeout*	等待超时，毫秒；
out	*pabtext*	文字信息缓冲区；
out	*puitextlen*	文字信息长度；
out	*pabwlt*	头像信息缓冲区；
out	*puiwltlen*	头像信息长度；
out	*pabfp*	指纹信息缓冲区；
out	*puifplen*	指纹信息长度；
out	*pabxtra*	附加信息缓冲区；
out	*puixtralen*	附加信息长度；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.5 mio_idcr_read_dn()

```
int32_t GHC_MIO_API mio_idcr_read_dn (
            size_t handle,
```

```
            uint32_t timeout,
            char * psdn )
```

```
#include <ghcmio_api.h>
```

读二代证DN码

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
out	*DN*码字符串，“|”分隔	

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.6 mio_idcr_read_samid()

```
int32_t GHC_MIO_API mio_idcr_read_samid (
            size_t handle,
            uint32_t timeout,
            char * pssamid )
```

```
#include <ghcmio_api.h>
```

读取SAM模块ID

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
out	*pssamid*	二代证SAM模块ID字符串；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.7 mio_idcr_read_uid()

```
int32_t GHC_MIO_API mio_idcr_read_uid (
            size_t handle,
            uint32_t timeout,
            char * psuid )
```

```
#include <ghcmio_api.h>
```

读取身份证物理ID

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
out	*psuid*	二代证物理ID字符串；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.8 mio_idcr_set_separation()

```
void GHC_MIO_API mio_idcr_set_separation (
            char separation )
```

```
#include <ghcmio_api.h>
```

设置读身份证输出文字信息时的分隔符，默认是'|'分隔，可以设置为其它任意单字节ASCII字符

参数

in	*separation*	分隔符；

返回

none

4.12.2.9 mio_idcr_transive()

```
int32_t GHC_MIO_API mio_idcr_transive (
            size_t handle,
            uint32_t timeout,
            uint8_t bCmd,
            uint8_t bArg,
            uint32_t * pdwRspCode,
            uint8_t * psRspData,
            int32_t iRspBufSize,
            int32_t * piRspDataLen )
```

```
#include <ghcmio_api.h>
```

收发AA指令

参数

in	*handle*,设备句柄；	
in	*timeout*,等待超时，毫秒；	
in	*bCmd*,指令字；	
in	*bArg*,指令参数；	
out	*pdwRspCode*,响应码；	
out	*psRspData*,应答数据；	
in	*iRspBufSize*,缓冲区长度；	
out	*piRspDataLen*,响应数据长度；	

返回

0 - 成功 <0 - 失败，详见错误代码表

4.12.2.10 mio_idcr_typeb_apdu()

```
int32_t GHC_MIO_API mio_idcr_typeb_apdu (
            size_t handle,
            uint32_t timeout,
            uint8_t * pscmd,
            uint16_t cmd_len,
            uint8_t * psrsp,
            uint16_t * p_rsp_len )
```

```
#include <ghcmio_api.h>
```

ISO14443 TypeB apdu 交互

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
in	*pscmd*	apdu指令；
in	*cmd_len*	apdu指令长度；
out	*psrsp*	rpdu应答接收缓冲区；
out	*p_rsp_len*	rpdu应答长度

返回

0 - 成功 <0 - 失败，详见错误代码表

4.13 RF125KHz_EMID_Card_Operation

125KHz射频卡操作接口

函数

- int32_t GHC_MIO_API mio_idcr_read_emid (size_t handle, uint32_t timeout, char *psemid)

  读取*125KHz EMID*卡号
- int32_t GHC_MIO_API mio_query_rf_card (size_t handle, char type, char *psuid)

  寻卡*TypeA/TypeB*
- int32_t GHC_MIO_API mio_probe_rf (size_t handle, uint8_t *in_out_feild, uint8_t *card_type)

  探测射频*TypeA/TypeB*卡片出入场,并识别入场卡片类型

4.13.1 详细描述

125KHz射频卡操作接口

4.13.2 函数说明

4.13.2.1 mio_idcr_read_emid()

```
int32_t GHC_MIO_API mio_idcr_read_emid (
            size_t handle,
            uint32_t timeout,
            char * psemid )
```

```
#include <ghcmio_api.h>
```

读取125KHz EMID卡号

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
out	*psemid*	EMID卡号字符串，12位长度，源数据6字节， 最后一字节（即字符串最后2位）为前5字节的校验值可忽略；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.13.2.2 mio_probe_rf()

```
int32_t GHC_MIO_API mio_probe_rf (
            size_t handle,
            uint8_t * in_out_feild,
            uint8_t * card_type )
```

```
#include <ghcmio_api.h>
```

探测射频TypeA/TypeB卡片出入场,并识别入场卡片类型

参数

in	*handle*	设备句柄；
in	*in_out_feild*	卡片进出场标识,2-卡片出场,3-卡片进场；
out	*card_type*	卡片类型, TYPE = 0x00：无卡 TYPE = 0x11：ISO14443A1-4 类型卡 TYPE = 0x13：ISO14443A1-3 Mifare S50(1K)卡 TYPE = 0x14：ISO14443A1-3 Mifare S70(4K)卡 TYPE = 0x15：ISO14443A1-3 Mifare Ultralight/Ntag20x 卡 TYPE = 0x18：ISO14443A1-3 Ntag21x 卡 TYPE = 0x1F：ISO14443A1-3 其它存储卡 TYPE = 0x21：ISO14443B1-4 类型卡 TYPE = 0x22：ISO14443B1-3 类型卡(如二代身份证卡)；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.13.2.3 mio_query_rf_card()

```
int32_t GHC_MIO_API mio_query_rf_card (
            size_t handle,
            char type,
            char * psuid )
```

```
#include <ghcmio_api.h>
```

寻卡TypeA/TypeB

参数

in	*handle*	设备句柄；
in	*type*	卡片类型，'A' - Type A, 'B' - Type B；
out	*psuid*	UID字符串，typeA卡4位长度(源数据2字节), typeB卡24位长度(源数据12字节)；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14 ISO14443A_Mifare_Card_Operation

Mifare TypeA射频卡操作接口

函数

- int32_t GHC_MIO_API mio_mifare_request (size_t handle, uint32_t timeout, char *cardtype, char *psuid)

  *Mifare*寻卡上电
- int32_t GHC_MIO_API mio_mifare_authentication (size_t handle, unsigned char sector, unsigned char key_type, const char *key)

  *Mifare*认证扇区（核对扇区密码）
- int32_t GHC_MIO_API mio_mifare_read_block (size_t handle, unsigned char block, char *pblockdat)

  *Mifare*读块数据
- int32_t GHC_MIO_API mio_mifare_write_block (size_t handle, unsigned char block, const char *pblockdat)

  *Mifare*写块数据
- int32_t GHC_MIO_API mio_mifare_init_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*将某一块初始化为钱包
- int32_t GHC_MIO_API mio_mifare_read_value (size_t handle, unsigned char block, int32_t *value)

  *Mifare*读钱包值
- int32_t GHC_MIO_API mio_mifare_increase_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*钱包充值
- int32_t GHC_MIO_API mio_mifare_decrease_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*钱包扣款
- int32_t GHC_MIO_API mio_mifare_restore (size_t handle, unsigned char block)

  *Mifare*数据回传（将*M1*卡某块的数据存储到卡内的缓存区）
- int32_t GHC_MIO_API mio_mifare_transfer (size_t handle, unsigned char block)

  *Mifare*数据传送（将*M1*卡内缓存区的数据写到某块）
- int32_t GHC_MIO_API mio_mifare_change_key (size_t handle, unsigned char sector, unsigned char key_type, const char *key)

  *Mifare*修改扇区密码，注意修改前要先认证扇区才可以

4.14.1 详细描述

Mifare TypeA射频卡操作接口

注意

Mifare卡有若干个扇区，扇区号从0开始递增，每扇区有4个块（编号0~3），某扇区某块的 绝对块号 计算方法为： block=扇区号*4+扇区块号（如0扇区0块的绝对块号为0*4+0=0；1扇区0块绝对块号为1*4+0=4）；

4.14.2 函数说明

4.14.2.1 mio_mifare_authentication()

```
int32_t GHC_MIO_API mio_mifare_authentication (
            size_t handle,
            unsigned char sector,
            unsigned char key_type,
            const char * key )
```

```
#include <ghcmio_api.h>
```

Mifare认证扇区（核对扇区密码）

参数

in	*handle*	设备句柄；
in	*sector*	要认证的扇区号；
in	*key_type*	秘钥类型，'A' - A秘钥；'B' - B秘钥；
in	*key*	秘钥数据，6个字节拆分后的12位字符串，如”FFFFFFFFFFFF”；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.2 mio_mifare_change_key()

```
int32_t GHC_MIO_API mio_mifare_change_key (
            size_t handle,
            unsigned char sector,
            unsigned char key_type,
            const char * key )
```

```
#include <ghcmio_api.h>
```

Mifare修改扇区密码，注意修改前要先认证扇区才可以

参数

in	*handle*	设备句柄；
in	*sector*	要认证的扇区号；
in	*key_type*	秘钥类型，'A' - A秘钥；'B' - B秘钥；
in	*key*	秘钥数据，6个字节拆分后的12位字符串，如”FFFFFFFFFFFF”；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.3 mio_mifare_decrease_value()

```
int32_t GHC_MIO_API mio_mifare_decrease_value (
            size_t handle,
            unsigned char block,
            int32_t value )
```

```
#include <ghcmio_api.h>
```

Mifare钱包扣款

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
in	*value*	扣款金额；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.4 mio_mifare_increase_value()

```
int32_t GHC_MIO_API mio_mifare_increase_value (
            size_t handle,
            unsigned char block,
            int32_t value )
```

```
#include <ghcmio_api.h>
```

Mifare钱包充值

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
in	*value*	充值金额；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.5 mio_mifare_init_value()

```
int32_t GHC_MIO_API mio_mifare_init_value (
            size_t handle,
            unsigned char block,
            int32_t value )
```

```
#include <ghcmio_api.h>
```

Mifare将某一块初始化为钱包

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
in	*value*	初始金额；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.6 mio_mifare_read_block()

```
int32_t GHC_MIO_API mio_mifare_read_block (
            size_t handle,
            unsigned char block,
            char * pblockdat )
```

```
#include <ghcmio_api.h>
```

Mifare读块数据

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
out	*pblockdat*	块数据，16个字节拆分后的32位字符串；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.7 mio_mifare_read_value()

```
int32_t GHC_MIO_API mio_mifare_read_value (
            size_t handle,
            unsigned char block,
            int32_t * value )
```

```
#include <ghcmio_api.h>
```

Mifare读钱包值

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
in	*value*	余额；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.8 mio_mifare_request()

```
int32_t GHC_MIO_API mio_mifare_request (
            size_t handle,
            uint32_t timeout,
            char * cardtype,
            char * psuid )
```

```
#include <ghcmio_api.h>
```

Mifare寻卡上电

参数

in	*handle*	设备句柄；
in	*timeout*	等待超时，毫秒；
out	*cardtype*	输出卡片类型，'A' - Type A, 'B' - Type B；
out	*psuid*	Mifare卡UID卡号字符串，8位长度，源数据4字节；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.9 mio_mifare_restore()

```
int32_t GHC_MIO_API mio_mifare_restore (
            size_t handle,
            unsigned char block )
```

```
#include <ghcmio_api.h>
```

Mifare数据回传（将M1卡某块的数据存储到卡内的缓存区）

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.10 mio_mifare_transfer()

```
int32_t GHC_MIO_API mio_mifare_transfer (
            size_t handle,
            unsigned char block )
```

```
#include <ghcmio_api.h>
```

Mifare数据传送（将M1卡内缓存区的数据写到某块）

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；

返回

0 - 成功 <0 - 失败，详见错误代码表

4.14.2.11 mio_mifare_write_block()

```
int32_t GHC_MIO_API mio_mifare_write_block (
            size_t handle,
            unsigned char block,
            const char * pblockdat )
```

```
#include <ghcmio_api.h>
```

Mifare写块数据

参数

in	*handle*	设备句柄；
in	*block*	绝对块号；
in	*pblockdat*	块数据，16个字节拆分后的32位字符串；

返回

0 - 成功 <0 - 失败，详见错误代码表

Chapter 5

文件说明

5.1 error_code.h 文件参考

Error code definitions.

枚举

- enum _TagErrCode {
  MIO_ERR_SUCCESS = 0, MIO_ERR_COM_OPEN_FAIL = -1, MIO_ERR_COM_CLOSE_FAIL = -2,
  MIO_ERR_COM_TIMEOUT = -3,
  MIO_ERR_COM_NO_ACK = -4, MIO_ERR_COM_XCHG_DATA = -5, MIO_ERR_ARG_TOO_SHORT = -6,
  MIO_ERR_ARG_TOO_LONG = -7,
  MIO_ERR_EEPROM_RW = -8, MIO_ERR_KEY_INDEX = -9, MIO_ERR_KEY_LENGTH = -10, MIO_ERR_INPUT_DATA
  = -11,
  MIO_ERR_KEY_NO_ZMK = -12, MIO_ERR_KEY_NO_ZPK = -13, MIO_ERR_KEY_NO_ZMK_ZPK = -14,
  MIO_ERR_KEY_VERIFY = -15,
  MIO_ERR_COUNT_GRAMA = -16, MIO_ERR_DATA_LEN_GRAMA = -17, MIO_ERR_PROC_CANCEL = -18,
  MIO_ERR_COM_WRONGNUM = -19,
  MIO_ERR_COM_HANDLE = -20, MIO_ERR_ARG_NULL_PTR = -21, MIO_ERR_DATA_CHECK_XOR = -22,
  MIO_ERR_NO_ENOUGH_MEM = -23,
  MIO_ERR_IC_POWEROFF_FAIL = -24, MIO_ERR_IC_CARDPRESENT_FAIL = -25, MIO_ERR_IC_APDU_FAIL
  = -26, MIO_ERR_IC_POWERON_FAIL = -27,
  MIO_ERR_PWD_NOT_GHC = -28, MIO_ERR_IC_DAT_NOT_FIND = -29, MIO_ERR_IC_PSE_DATA = -30,
  MIO_ERR_IC_FIND_AID = -31,
  MIO_ERR_IC_SELECT_AID = -32, MIO_ERR_IC_GPO = -33, MIO_ERR_IC_NO_ALLINFO_REC = -34,
  MIO_ERR_IC_NO_ALLINFO_GETDATA = -35,
  MIO_ERR_IC_NO_THIS_TAG = -36, MIO_ERR_IC_APDU_UNSUPPORT = -37, MIO_ERR_ARQC_DATAFIELD
  = -38, MIO_ERR_ARQC_IN_TRANSDATA = -39,
  MIO_ERR_ARQC_FAIL = -40, MIO_ERR_FIELD55_DATA = -41, MIO_ERR_TRANSDETAIL_FORMAT = -42,
  MIO_ERR_TRANSDETAIL_REC = -43,
  MIO_ERR_TRANSDETAIL_REC_DATA = -44, MIO_ERR_ARPC_NOT_FOUND = -45, MIO_ERR_SCRPTTEMP_NOT_FOUND
  = -46, MIO_ERR_SCRPT_NOT_FOUND = -47,
  MIO_ERR_SCRPTMARK_NOT_FOUND = -48, MIO_ERR_EXTAUTH_FAIL = -49, MIO_ERR_SCRPT_EXEC_FAIL
  = -50, MIO_ERR_DATA_FORMAT_REC = -51,
  MIO_ERR_BUF_NOT_ENOUGH = -52, MIO_ERR_LOADDETAIL_FORMAT = -53, MIO_ERR_LOADDETAIL_REC
  = -54, MIO_ERR_LOADDETAIL_REC_DATA = -55,
  MIO_ERR_READ_VER_FAIL = -56, MIO_ERR_9000_FAIL = -57, MIO_ERR_STATE_FAIL = -58, MIO_ERR_EXEC_FAIL
  = -59,
  MIO_ERR_PHOTODIR = -101, MIO_ERR_WRITE_WLT = -102, MIO_ERR_MAKEPHOTO = -103,
  MIO_ERR_LOADPHOTO = -104,
  MIO_ERR_LOADLIB = -106, MIO_ERR_LOADFUNC = -107, MIO_ERR_FINDID = -111, MIO_ERR_MSG_DATA_NULL
  = -200,
  MIO_ERR_MSG_DATA_TOOLONG = -201, MIO_ERR_MSG_WRITEFAIL = -202 }

5.1.1 详细描述

Error code definitions.

作者

loongworks@163.com

版本

V1.20.1016.1

日期

14-July-2020

5.2 ghcmio_api.h 文件参考

Export ghcmio library APIs.

```
#include "stdint.h"
#include "stddef.h"
```

宏定义

- #define GHC_MIO_API
  声明*Windows*下的函数调用约定.
- #define IDCR_READ_MODE_COMPATIBLE 0x0800u
  二代证读卡（*USB*）采用兼容模式，以支持速度较慢的*PC*或云桌面*(*除非实际测试时，频繁遇到*USB*读身份证信息失败的情况，不建议采用此方式，因会大幅降低读卡速度*)*.
- #define IDCR_READ_MODE_EXT 0x0020u
  二代证读卡，读取附加信息.
- #define IDCR_READ_MODE_REREAD 0x0010u
  二代证读卡，强制重新读取.
- #define IDCR_READ_MODE_COMPOSITE 0x0008u
  二代证读卡，合成正反面照片.
- #define IDCR_READ_MODE_PHOTO 0x0004u
  二代证读卡，读取头像.
- #define IDCR_READ_MODE_TEXT 0x0002u
  二代证读卡，读取文本.
- #define IDCR_READ_MODE_FINGER 0x0001u
  二代证读卡，读取指纹信息.

类型定义

- typedef void(CALLBACK * cb_on_swipe_magstripe) (int code, char *info)
  读磁卡信息回调函数

函数

- void GHC_MIO_API mio_lib_version (char *version_info)
  获取动态库版本信息（不操作硬件）
- void GHC_MIO_API mio_get_log_path (char *path_log)
  获取日志路径
- int32_t GHC_MIO_API mio_firm_version (size_t handle, uint8_t *pbsVerData, int32_t *piVerLen)
  读取读卡器固件版本
- int32_t GHC_MIO_API mio_beep (size_t handle, uint8_t duration, uint8_t interval, uint8_t beeptimes)
  蜂鸣控制
- int32_t GHC_MIO_API mio_enumerate (char *devs_list)
  枚举主机上的串口和可用的*USB*设备
- int32_t GHC_MIO_API mio_enum_devices (int32_t dev_type, char **p_out_list)
  枚举主机上的串口和可用的*USB*设备
- void GHC_MIO_API mio_free_list (const char *p_list)
  释放*mio_enum_devices*生成的设备列表
- size_t GHC_MIO_API mio_open (const char *psport, char *psexport, int32_t baud)
  打开一个设备
- int32_t GHC_MIO_API mio_close (size_t handle, int32_t iPortTypeRfu)
  关闭一个设备
- int32_t GHC_MIO_API mio_set_baud (size_t handle, uint32_t baud, uint8_t save_mode)
  设置波特率
- size_t GHC_MIO_API mio_open_and_set_baud (const char *psport, char *psexport, int32_t origin_baud, int32_t dest_baud)
  打开一个串口设备并设置波特率, 支持*9600, 19200, 38400, 57600, 115200*
- int32_t GHC_MIO_API mio_reset_device (size_t handle, uint8_t bMute)
  复位设备, 已经建立的连接将丢失重置, 需重新打开设备
- int32_t GHC_MIO_API mio_msr_polling_start (size_t handle, int32_t query_interval, cb_on_swipe_magstripe cb)
  启动一个轮询读磁卡的线程,获取到磁条信息时或异常时将通过回调函数反馈
- void GHC_MIO_API mio_msr_polling_interval (int32_t interval_ms)
  设置轮询间隔时间,立即生效
- void GHC_MIO_API mio_msr_polling_stop (void)
  终止轮询读磁卡的线程
- int32_t GHC_MIO_API mio_msr_qm_start (size_t handle)
  启动读磁卡
- int32_t GHC_MIO_API mio_msr_qm_status (size_t handle)
  获取读磁卡状态
- int32_t GHC_MIO_API mio_msr_qm_get (size_t handle, char *track1, char *track2, char *track3, int32_t *track1len, int32_t *track2len, int32_t *track3len)
  获取读到的磁条数据
- int32_t GHC_MIO_API mio_msr_qm_stop (size_t handle)
  结束读卡
- int32_t GHC_MIO_API mio_msr_qm_status_raw (size_t handle)
  获取读磁卡*raw*数据状态
- int32_t GHC_MIO_API mio_msr_qm_get_raw (size_t handle, int32_t index, char *track, int32_t *tracklen)
  获取读到的磁条数据
- int32_t GHC_MIO_API mio_msr_qm_stop_raw (size_t handle)
  结束*raw*读卡
- int32_t GHC_MIO_API mio_msr_raw2str (char *praw, int32_t raw_len, char *pstr)
  将磁条*raw*数据转为字符串
- int32_t GHC_MIO_API mio_msr_get_crypto_card_sn (char *praw, int32_t raw_len, char *pstr)

将磁条*raw*数据转为卡号

- int32_t GHC_MIO_API mio_msr_cancel (size_t handle)

  取消读写磁条

- int32_t GHC_MIO_API mio_msr_erase (size_t handle, int32_t trackselect, int32_t timeout_ms)

  擦除磁条信息

- int32_t GHC_MIO_API mio_msr_read (size_t handle, int32_t trackselect, char *track1, char *track2, char *track3, int32_t timeout)

  读取磁条信息【阻塞方式】

- int32_t GHC_MIO_API mio_msr_write (size_t handle, int32_t trackselect, int32_t delimiter, const char *track1, const char *track2, const char *track3, int32_t timeout_ms)

  写磁条信息

- int32_t GHC_MIO_API mio_icc_poweron (size_t handle, uint8_t card_type, uint16_t timeout, char *atr, int32_t *length)

  给设备上电

- int32_t GHC_MIO_API mio_icc_poweroff (size_t handle, uint8_t card_type, uint16_t timeout)

  设备下电

- int32_t GHC_MIO_API mio_icc_status (size_t handle, uint8_t card_type)

  获取接触卡状态

- int32_t GHC_MIO_API mio_icc_apdu (size_t handle, uint8_t card_type, const char *psapdu, int32_t apdu_length, char *psrpdu, int32_t *rpdu_len)

  发送*APDU*命令

- int32_t GHC_MIO_API **mio_etc_read_number** (size_t handle, uint8_t card_slot, uint16_t timeout, char *pscard_number)
- int32_t GHC_MIO_API mio_pboc_get_card_info (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, const char *psTagList, uint8_t *pbsCardType, uint8_t *psUserInfo, uint16_t *wInfoLen, uint8_t bXtraProc)

  金融*IC*卡 获取卡片信息

- int32_t GHC_MIO_API **mio_pboc_get_arqc** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, uint16_t wAidLen, const uint8_t *psTxDat, uint16_t wTxDatLen, uint8_t *psArqc, uint16_t *pwArqcLen, uint8_t bIsNeedPwrOn)
- int32_t GHC_MIO_API **mio_pboc_arpc_exec_script** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const uint8_t *psTxDat, uint16_t wTxDatLen, const uint8_t *psArpc, uint16_t wArpcLen, uint8_t *pbsOutBuf, uint16_t *pwOutLen)
- int32_t GHC_MIO_API **mio_pboc_get_trans_detail** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, uint8_t num, const char *aid, uint8_t *psDetail, uint16_t *pwDetailLen, uint8_t bIsNeedPwrOn)
- int32_t GHC_MIO_API **mio_pboc_get_card_info_all** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *psTagList, uint8_t *pbsCardType, uint8_t *pbsOutBuf, uint16_t *pwOutBufLen, uint8_t bIsNeedPwrOn)
- int32_t **mio_pboc_is_app_locked** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, const char *aid, uint8_t bIsNeedPwrOn)
- int16_t GHC_MIO_API **mio_pboc_get_load_detail** (size_t handle, uint8_t bCardSlot, uint16_t wTimeout, uint8_t num, const char *aid, uint8_t *psDetail, uint16_t *pwDetailLen, uint8_t bIsNeedPwrOn)
- int32_t GHC_MIO_API mio_idcr_read_bin (size_t handle, uint32_t read_mode, uint32_t timeout, uint8_t *pabtext, uint32_t *puitextlen, uint8_t *pabwlt, uint32_t *puiwltlen, uint8_t *pabfp, uint32_t *puifplen, uint8_t *pabxtra, uint32_t *puixtralen)

  读取身份证原始数据

- int32_t GHC_MIO_API mio_idcr_read (size_t handle, uint32_t read_mode, uint32_t timeout)

  读身份证并缓存信息

- int32_t GHC_MIO_API mio_idcr_getinfo (size_t handle, uint32_t read_mode, uint32_t timeout, int32_t pictype, char *pstxt, char *psfp, const char *psphotopath, const char *psfrontpath, const char *psbackpath)

  读取身份证信息

- void GHC_MIO_API mio_idcr_set_separation (char separation)

  设置读身份证输出文字信息时的分隔符，默认是'|'分隔，可以设置为其它任意单字节*ASCII*字符

- int32_t GHC_MIO_API mio_idcr_getraw (size_t handle, uint32_t raw_type, char *psbuf, int32_t *pidatlen)

  获取身份证未解码的原始数据

- int32_t GHC_MIO_API mio_idcr_read_samid (size_t handle, uint32_t timeout, char *pssamid)

  读取*SAM*模块*ID*
- int32_t GHC_MIO_API mio_idcr_transive (size_t handle, uint32_t timeout, uint8_t bCmd, uint8_t bArg, uint32_t *pdwRspCode, uint8_t *psRspData, int32_t iRspBufSize, int32_t *piRspDataLen)

  收发*AA*指令
- int32_t GHC_MIO_API mio_idcr_read_uid (size_t handle, uint32_t timeout, char *psuid)

  读取身份证物理*ID*
- int32_t GHC_MIO_API mio_idcr_read_dn (size_t handle, uint32_t timeout, char *psdn)

  读二代证*DN*码
- int32_t GHC_MIO_API mio_idcr_typeb_apdu (size_t handle, uint32_t timeout, uint8_t *pscmd, uint16_t cmd_len, uint8_t *psrsp, uint16_t *p_rsp_len)

  ISO14443 TypeB apdu 交互
- int32_t GHC_MIO_API mio_idcr_read_emid (size_t handle, uint32_t timeout, char *psemid)

  读取*125KHz EMID*卡号
- int32_t GHC_MIO_API mio_query_rf_card (size_t handle, char type, char *psuid)

  寻卡*TypeA/TypeB*
- int32_t GHC_MIO_API mio_probe_rf (size_t handle, uint8_t *in_out_feild, uint8_t *card_type)

  探测射频*TypeA/TypeB*卡片出入场,并识别入场卡片类型
- int32_t GHC_MIO_API mio_mifare_request (size_t handle, uint32_t timeout, char *cardtype, char *psuid)

  *Mifare*寻卡上电
- int32_t GHC_MIO_API mio_mifare_authentication (size_t handle, unsigned char sector, unsigned char key_type, const char *key)

  *Mifare*认证扇区（核对扇区密码）
- int32_t GHC_MIO_API mio_mifare_read_block (size_t handle, unsigned char block, char *pblockdat)

  *Mifare*读块数据
- int32_t GHC_MIO_API mio_mifare_write_block (size_t handle, unsigned char block, const char *pblockdat)

  *Mifare*写块数据
- int32_t GHC_MIO_API mio_mifare_init_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*将某一块初始化为钱包
- int32_t GHC_MIO_API mio_mifare_read_value (size_t handle, unsigned char block, int32_t *value)

  *Mifare*读钱包值
- int32_t GHC_MIO_API mio_mifare_increase_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*钱包充值
- int32_t GHC_MIO_API mio_mifare_decrease_value (size_t handle, unsigned char block, int32_t value)

  *Mifare*钱包扣款
- int32_t GHC_MIO_API mio_mifare_restore (size_t handle, unsigned char block)

  *Mifare*数据回传（将*M1*卡某块的数据存储到卡内的缓存区）
- int32_t GHC_MIO_API mio_mifare_transfer (size_t handle, unsigned char block)

  *Mifare*数据传送（将*M1*卡内缓存区的数据写到某块）
- int32_t GHC_MIO_API mio_mifare_change_key (size_t handle, unsigned char sector, unsigned char key_type, const char *key)

  *Mifare*修改扇区密码，注意修改前要先认证扇区才可以

5.2.1 详细描述

Export ghcmio library APIs.

作者

loongworks@163.com

版本

V1.20.713.2

日期

14-July-2020

Index