Advanced Card Systems Ltd.
Card & Reader Technologies

# ACR1281S-C1

# 双界面读写器

# （串口）

应用程序编程接口 V1.01



info@acs.com.hk
**www.acs.com.hk**

# 目录



# 表目录

# 1.0. 简介

本手册介绍了如何使用 ACR1281S-C1 软件接口进行 ACR1281S-C1 读写器应用程序的开发。此软件接口以 32 位和 64 位动态链接库（DLL）的形式提供，可以通过当前流行的开发工具进行编程，其中包括 Java、Delphi、Visual Basic、Visual C++、Visual C#和 Visual Basic .Net。

ACR1281S-C1 的 DLL 是一套供应用软件使用的高阶函数。它提供了一致的 API（应用程序编程接口）供各个程序调用，既可以操作 ACR1281S-C1，也可以操作相应的卡片。DLL 通过操作系统提供的通信端口与 ACR1281S-C1 进行通信。

ACR1281S-C1 API 定义了访问 ACR1281S-C1 的通用方式。应用程序可以通过接口函数启用 ACR1281S-C1 并对出示的卡片进行操作。

我们提供了头文件“acr.h”供程序开发人员使用，其中包含此文档介绍的所有函数原型和宏。

## 1.1. 特性

- RS232串行接口：波特率 = 9.6 kbps（默认）、19.2 kbps、38.4 kbps、57.6 kbps、115.2 kbps、230.4 kbps
- USB接口取电
- 仿CCID架构（二进制格式）
- 非接触式智能卡读写器：
  - 读写速度达 848 kbps
  - 内置天线用于读写非接触式标签，读取智能卡的距离可达 50 mm（视标签的类型而定）
  - 支持 ISO 14443 第 4 部分的 A 类和 B 类卡，以及 Mifare 系列卡
  - 内建防冲突特性（任何时候都只能访问 1 张标签）
  - 支持扩展的 APDU（最大 64K 字节）
- 接触式智能卡读写器：
  - 支持 ISO 7816 的 A 类、B 类和 C 类（5V、3V、1.8V）卡
  - 支持符合 T=0 或 T=1 协议的微处理器卡
  - 支持各类存储卡
  - 符合 ISO7816 标准的 SAM 卡槽
- 内置外围设备：
  - 2 个用户可控的 LED 指示灯
  - 1 个用户可控的蜂鸣器
- 具有USB固件升级能力
- 符合下列标准：
  - ISO 14443
  - ISO 7816
  - CE
  - FCC
  - RoHS

# 2.0. ACR API

## 2.1. 使用 ACR API

使用 ACR API 之前，首先要在头文件“acr.h”中包含源代码。

**源代码示例：**

```
#include <stdio.h>
#include <acr.h>

int main(int argc, char *argv[])
{
   HANDLE hReader;
   DWORD ret;

   // Open reader using COM1
   ret = ACR_Open(TEXT("\\\\.\\COM1"), &hReader);
   if (ret != ERROR_SUCCESS)
   {
      printf("ACR_Open failed with error 0x%08X\n", ret);
      return 1;
   }

   // TODO:Place other API function calls here
   // ...

   // Close reader
   ret = ACR_Close(hReader);
   if (ret != ERROR_SUCCESS)
      printf("ACR_Close failed with error 0x%08X\n", ret);

   return 0;
}
```

### 2.1.1. 打开读写器

在 Windows 平台下，端口名称定义为"\\.\COMx"，其中 x 表示端口号。例如"\\.\COM1"表示第一个串行端口。

在 Linux 平台下，端口名称可以是"/dev"目录下的任意文件。由标准的 Linux 串口驱动创建的设备文件为"/dev/ttySx"，其中 x 表示端口号。例如，"/dev/ttyS0"是指第一个串行端口。

如果成功打开端口，`ACR_Open()`会在参数中返回一个读写器操作句柄。

### 2.1.2. 关闭读写器

完成以上操作后，可以通过 `ACR_Close()`来关闭读写器。

### 2.1.3. 使用 ANSI 和 Unicode 函数

若源代码定义了 UNICODE 宏，则下列函数会转换为 Unicode 版本。

```
ACR_Open()-> ACR_OpenW()
```

否则，会转换为 ANSI 版本。

ACR_Open()-> ACR_OpenA()

## 2.2. 读写器

### 2.2.1. 预定义文档

#### 2.2.1.1. ACR_Open 和 ACR_OpenA

**格式：**

```
#define ACR_Open ACR_OpenA
```

如果定义了 UNICODE，ACR_Open 会被转换为 ACR_OpenW()函数。

否则，会转换为 ACR_OpenA()函数。

### 2.2.2. 函数文档

#### 2.2.2.1. ACR_OpenA

**格式：**

```
DWORD WINAPI ACR_OpenA        ( LPCSTR          portName,
                                LPHANDLE        phReader
                              )
```

**函数描述：打开读写器（ANSI）。**

此函数用于打开读写器并返回一个句柄值作为参考。

| 参数 | 说明 | |
|---|---|---|
| [in] portName | 端口名称。<br><br>在 Windows 平台下，端口名称定义为“\\.\COMx”，其中 x 是端口号。例如，“\\.\COM1”表示第一个串行端口。<br><br>在 Linux 平台下，端口名称可以是“/dev”目录下的任意文件。由标准的 Linux 串口驱动创建的设备文件为“/dev/ttySx”，其中 x 表示端口号。例如，“/dev/ttyS0” 是指第一个串行端口。 | |
| [out] phReader | 指向 HANDLE 变量的指针。 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表1：** ACR_OpenA 函数描述

**源代码示例：**

```
HANDLE hReader;
DWORD ret;

// Open reader using COM1
ret = ACR_Open(TEXT("\\\\.\\COM1"), &hReader);
if (ret != ERROR_SUCCESS)
   printf("ACR_Open failed with error 0x%08X\n", ret);
```

### 2.2.2.2. ACR_OpenW

**格式：**

```
DWORD WINAPI ACR_OpenW        ( LPCWSTR          portName,
                                LPHANDLE         phReader
                              )
```

**函数描述：打开读写器（Unicode）。**

此函数用于打开读写器并返回一个句柄值作为参考。

| 参数 | 说明 | |
|---|---|---|
| [in] portName | 端口名称。<br><br>在 Windows 平台下，端口名称定义为“\\.\COMx”，其中 x 是端口号。例如，“\\.\COM1”表示第一个串行端口。<br><br>在 Linux 平台下，端口名称可以是“/dev”目录下的任意文件。由标准的 Linux 串口驱动创建的设备文件为“/dev/ttySx”，其中 x 表示端口号。例如，“/dev/ttyS0” 是指第一个串行端口。 | |
| [out] phReader | 指向 HANDLE 变量的指针。 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表2**：ACR_OpenW 函数描述

**源代码示例：**

```
HANDLE hReader;
DWORD ret;

// Open reader using COM1
ret = ACR_Open(TEXT("\\\\.\\COM1"), &hReader);
if (ret != ERROR_SUCCESS)
   printf("ACR_Open failed with error 0x%08X\n", ret);
```

### 2.2.2.3. ACR_Close

格式：

```
DWORD WINAPI ACR_Close        ( HANDLE     hReader )
```

**函数描述：关闭读写器。**

此函数用于关闭读写器并释放资源。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表3**：ACR_Close 函数描述

**源代码示例：**

```
DWORD ret;

// Close reader
ret = ACR_Close(hReader);
if (ret != ERROR_SUCCESS)
   printf("ACR_Close failed with error 0x%08X\n", ret);
```

### 2.2.2.4. ACR_GetNumSlots

格式：

```
DWORD WINAPI ACR_GetNumSlots  ( HANDLE     hReader,
                                LPDWORD    pNumSlots
                              )
```

**函数描述：获取卡槽号。**

此函数用于获取读写器的卡槽编号。

| 参数 | | 说明 |
|---|---|---|
| [in] hReader | | ACR_Open()函数返回的参考值。 |
| [out] pNumSlots | | 指向一个 DWORD 变量的指针，该变量用于接收返回的卡槽号。 |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表4：** ACR_GetNumSlots 函数描述

**源代码示例：**

```
DWORD numSlots;
DWORD ret;

// Get number of slots
ret = ACR_GetNumSlots (hReader, &numSlots);
if (ret != ERROR_SUCCESS)
   printf("ACR_GetNumSlots failed with error 0x%08X\n", ret);
```

### 2.2.2.5. ACR_GetBaudRate

格式：

```
DWORD WINAPI ACR_GetBaudRate ( HANDLE        hReader,
                               LPDWORD       pBaudRate
                             )
```

**函数描述：获取波特率。**

此函数用于获取读写器的通信波特率。

| 参数 | | 说明 |
|---|---|---|
| [in] hReader | | ACR_Open()函数返回的参考值。 |
| [out] pBaudRate | | 指向一个 DWORD 变量的指针，该变量用于接收返回的通信波特率。 |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |

| 参数 | | 说明 |
|---|---|---|
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表5**：ACR_GetBaudRate 函数描述

**源代码示例：**

```
DWORD baudRate;
DWORD ret;

// Get baud rate
ret = ACR_GetBaudRate(hReader, &baudRate);
if (ret != ERROR_SUCCESS)
   printf("ACR_GetBaudRate failed with error 0x%08X\n", ret);
```

### 2.2.2.6. ACR_SetBaudRate

**格式：**

```
DWORD WINAPI ACR_SetBaudRate  ( HANDLE          hReader,
                                DWORD           baudRate
                              )
```

**函数描述：设定波特率。**

此函数用于设定读写器的通信波特率。ACR1281S-C1 读写器支持 9600、19200、38400、57600、115200、128000、230400、250000 和 500000 bps 的通信波特率。

| 参数 | | 说明 |
|---|---|---|
| [in] hReader | | ACR_Open()函数返回的参考值。 |
| [in] baudRate | | 波特率。 |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表6**：ACR_SetBaudRate 函数描述

**源代码示例：**

```
DWORD ret;

// Set baud rate to 115200 bps
ret = ACR_SetBaudRate(hReader, 115200);
if (ret != ERROR_SUCCESS)
   printf("ACR_SetBaudRate failed with error 0x%08X\n", ret);
```

### 2.2.2.7. ACR_GetTimeouts

**格式：**

```
DWORD WINAPI ACR_GetTimeouts ( HANDLE          hReader,
                               PACR_TIMEOUTS   pTimeouts
                             )
```

**函数描述：获取超时时间。**

此函数用于获得读写器的状态操作与响应操作的超时参数。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [out] pTimeouts | 指向一个 ACR_TIMEOUTS 结构体的指针，该结构体用于接收返回的超时信息。 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表7**：ACR_GetTimeouts 函数描述

**源代码示例：**

```
ACR_TIMEOUTS timeouts;
DWORD ret;

// Get timeouts
ret = ACR_GetTimeouts (hReader, &timeouts);
if (ret != ERROR_SUCCESS)
   printf("ACR_GetTimeouts failed with error 0x%08X\n", ret);
```

### 2.2.2.8. ACR_SetTimeouts

**格式：**

```
DWORD WINAPI ACR_SetTimeouts ( HANDLE                hReader,
                               const PACR_TIMEOUTS   pTimeouts
                             )
```

**函数描述：设定超时时间。**

此函数用于设定读写器的状态操作与响应操作的超时参数。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [in] pTimeouts | 指向一个 ACR_TIMEOUTS 结构体的参数，该结构体中包含了新的超时值。 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表8：** ACR_SetTimeouts 函数描述

**源代码示例：**

```
ACR_TIMEOUTS timeouts;
DWORD ret;

// Get timeouts
// ...

// Modify status timeout to 100 ms
timeouts.statusTimeout = 100;

// Set timeouts
ret = ACR_SetTimeouts(hReader, &timeouts);
if (ret != ERROR_SUCCESS)
   printf("ACR_SetTimeouts failed with error 0x%08X\n", ret);
```

## 2.3. 卡片

### 2.3.1. 函数文档

#### 2.3.1.1. ACR_ExchangeApdu

**格式：**

```
DWORD WINAPI ACR_ExchangeApdu ( HANDLE          hReader,
                                DWORD           slotNum,
                                const LPBYTE    sendBuffer,
                                DWORD           sendBufferLen,
                                LPBYTE          recvBuffer,
                                LPDWORD         pRecvBufferLen
                              )
```

**函数描述：交换 APDU。**

此函数用于将 APDU 命令发送到卡片，以及从卡片接收 APDU 响应。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [in] slotNum | 卡槽编号。 | |
| [in] sendBuffer | 指向实际要写入卡片的数据的指针。 | |
| [in] sendBufferLen | sendBuffer 参数的长度（字节数）。 | |
| [out] recvBuffer | 指向卡片返回的数据的指针。 | |
| [in,out] pRecvBufferLen | recvBuffer 参数的长度（字节数），并且接收实际从卡片收到的字节数 | |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表9**：ACR_ExchangeAPDU 函数描述

**源代码示例：**

```
BYTE command[] = { 0x80, 0x84, 0x00, 0x00, 0x08 };
DWORD commandLen = sizeof(command);
BYTE response[300];
DWORD responseLen = sizeof(response);
DWORD ret;

// Exchange APDU in slot 0
ret = ACR_ExchangeApdu(hReader, 0, command, commandLen, response,
&responseLen);
if (ret != ERROR_SUCCESS)
   printf("ACR_ExchangeApdu failed with error 0x%08X\n", ret);
```

### 2.3.1.2. ACR_GetIccStatus

**格式：**

```
DWORD WINAPI ACR_GetIccStatus( HANDLE          hReader,
                               DWORD           slotNum,
                               LPDWORD         pIccStatus
                             )
```

**函数描述：获取 ICC 的状态。**

此函数用于返回读写器卡槽的当前 ICC 状态。

| 参数 | 说明 |
|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 |
| [in] slotNum | 卡槽编号。 |
| [out] pIccStatus | 指向一个 DWORD 变量的指针，该变量用于接收返回的 ICC 状态。<br>值 — 含义<br>ACR_ICC_UNKNOWN — 当前处于未知状态。<br>ACR_ICC_ABSENT — 读写器中没有卡片。<br>ACR_ICC_PRESENT — 读写器中有卡片<br>ACR_ICC_POWERED — 正在向卡片供电。 |
| **返回值** | ERROR_SUCCESS 操作成功完成。 |

| 参数 | | 说明 |
|---|---|---|
| | | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件 “errno.h”以及 ACR 错误代码集。 |
| | Failure | |

**表10 ：** ACR_GetIccStatus 函数描述

**源代码示例：**

```
DWORD iccStatus;
DWORD ret;

// Get ICC status in slot 0
ret = ACR_GetIccStatus(hReader, 0, &iccStatus);
if (ret != ERROR_SUCCESS)
   printf("ACR_GetIccStatus failed with error 0x%08X\n", ret);
```

### 2.3.1.3. ACR_PowerOffIcc

**格式：**

```
DWORD WINAPI ACR_PowerOffIcc  ( HANDLE          hReader,
                                DWORD           slotNum
                              )
```

**函数描述：关闭卡槽中 ICC 的电源。**

此函数用于对卡槽中的 ICC 进行下电。

| 参数 | | 说明 |
|---|---|---|
| [in] hReader | | ACR_Open()函数返回的参考值。 |
| [in] slotNum | | 卡槽编号。 |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件 “errno.h”以及 ACR 错误代码集。 |

**表11 ：** ACR_PowerOffIcc 函数描述

**源代码示例：**

```
DWORD ret;

// Power off slot 0
ret = ACR_PowerOffIcc(hReader, 0);
if (ret != ERROR_SUCCESS)
   printf("ACR_PowerOffIcc failed with error 0x%08X\n", ret);
```

### 2.3.1.4. ACR_PowerOnIcc

**格式：**

```
DWORD WINAPI ACR_PowerOnIcc   ( HANDLE          hReader,
                                DWORD           slotNum,
                                LPBYTE          atr,
                                LPDWORD         pAtrLen
                              )
```

**函数描述：为卡槽中的 ICC 上电。**

此函数用于对卡槽中的卡片进行上电，同时返回卡片的 ATR 字符串。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [in] slotNum | 卡槽编号。 | |
| [out] atr | 指向缓冲区的指针，该缓冲区用于接收从卡片返回的 ATR 字符串。 | |
| [in,out] pAtrLen | atr 参数的长度（字节数），并且接收实际从卡片收到的字节数 | |
| **返回值** | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表12 ：** ACR_PowerOnICC 函数描述

**源代码示例：**

```
BYTE atr[64];
DWORD atrLen = sizeof(atr);
DWORD ret;

// Power on slot 0
ret = ACR_PowerOnIcc(hReader, 0, atr, &atrLen);
if (ret != ERROR_SUCCESS)
   printf("ACR_PowerOnIcc failed with error 0x%08X\n", ret);
```

### 2.3.1.5. ACR_SetProtocol

**格式：**

```
DWORD WINAPI ACR_SetProtocol  ( HANDLE          hReader,
                                DWORD           slotNum,
                                DWORD           preferredProtocols,
                                LPDWORD         pActiveProtocol
                              )
```

**函数描述：设置协议。**

此函数用于根据卡片返回的 ATR 字符串选择协议和参数。设定协议前，首先必须通过 ACR_PowerOnIcc()函数对卡片进行上电。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [in] slotNum | 卡槽编号。 | |
| [in] preferredProtocols | 可接受协议的位掩码。可能的值也可以与 OR 操作相结合。 | |
| | **值** | **含义** |
| | ACR_PROTOCOL_T0 | T=0 是可接受的协议。 |
| | ACR_PROTOCOL_T1 | T=1 是可接受的协议。 |

| 参数 | 说明 | |
|---|---|---|
| [out] pActiveProtocol | 标记已经建立的活动协议 | |
| | 值 | 含义 |
| | ACR_PROTOCOL_T0 | T=0 是活动协议。 |
| | ACR_PROTOCOL_T1 | T=1 是活动协议。 |
| | ACR_PROTOCOL_UNDEFINED | 协议尚未定义。 |
| 返回值 | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br>对于 Linux 平台，请参阅 GNU C 库头文件 “errno.h”以及 ACR 错误代码集。 |

**表13 ：ACR_SetProtocol 函数描述**

**源代码示例：**

```
DWORD activeProtocol;
DWORD ret;

// Power on slot 0
// ...

// Set protocol to T=0 or T=1
ret = ACR_SetProtocol(hReader, 0, ACR_PROTOCOL_T0 | ACR_PROTOCOL_T1,
&activeProtocol);
if (ret != ERROR_SUCCESS)
   printf("ACR_SetProtocol failed with error 0x%08X\n", ret);
```

## 2.4. 外围设备

### 2.4.1. 函数文档

#### 2.4.1.1. ACR_Control

**格式：**

```
DWORD WINAPI ACR_Control      ( HANDLE          hReader,
                               DWORD           slotNum,
                               DWORD           controlCode,
                               LPCVOID         inBuffer,
```

```
    DWORD           inBufferSize,
    LPVOID          outBuffer,
    DWORD           outBufferSize,
    LPDWORD         pBytesReturned
)
```

**函数描述：控制外围设备。**

此函数可以让用户直接控制读写器的外围设备。

| 参数 | 说明 | |
|---|---|---|
| [in] hReader | ACR_Open()函数返回的参考值。 | |
| [in] slotNum | 卡槽编号。 | |
| [in] controlCode | 操作的控制码。该值用于指定要进行的特定操作。 | |
| [in] inBuffer | 指向一个缓冲区，该缓冲区内存放了执行操作所需的数据。若 controlCode 参数指定的操作不需要输入数据，则此参数可以为 NULL。 | |
| [in] inBufferSize | inBuffer 参数所指向缓冲区的大小（字节数） | |
| [out] outBuffer | 指向一个缓冲区，该缓冲区用于接收操作的响应数据。若 controlCode 参数指定的操作不会产生响应数据，则此参数可以为 NULL。 | |
| [in] outBufferSize | outBuffer 参数所指向缓冲区的大小（字节数） | |
| [out] pBytesReturned | 指向一个 DWORD，该 DWORD 用于接收存储在 outBuffer 参数指定的缓冲区内的数据的大小（以字节为单位）。 | |
| **返回值** | ERROR_SUCCESS | 操作成功完成。 |
| | Failure | 错误代码。<br><br>对于 Windows 平台，请参阅 Windows API 和 ACR 的错误代码集。<br><br>对于 Linux 平台，请参阅 GNU C 库头文件“errno.h”以及 ACR 错误代码集。 |

**表14** ：ACR_Control 函数描述

**源代码示例：**

```
BYTE command[] = { 0xE0, 0x00, 0x00, 0x18, 0x00 };
DWORD commandLen = sizeof(command);
BYTE response[20];
DWORD responseLen = sizeof(response);
DWORD ret;

// Get firmware version
ret = ACR_Control(hReader, 0, IOCTL_ACR_CCID_ESCAPE, command,
commandLen, response, responseLen, &responseLen);
if (ret != ERROR_SUCCESS)
   printf("ACR_Control failed with error 0x%08X\n", ret);
```

# Appendix A. 数据结构

## 附录 A.1. _ACR_TIMEOUTS 结构引用

_ACR_TIMEOUTS 结构主要用于 ACR_GetTimeouts() 和 ACR_SetTimeouts() 函数

1. DWORD statusTimeout
   - 状态超时值，单位为毫秒
   - 默认为 2000 毫秒
2. DWORD numStatusRetries
   - 状态重试次数
   - 默认为 1 次
3. DWORD responseTimeout
   - 响应超时值，单位为毫秒
   - 默认为 10000 毫秒
4. DWORD numResponseRetries
   - 响应重试次数
   - 默认为 1 次

# Appendix B. acr.h 文件引用

## Appendix B.1. 数据结构

1. struct _ACR_TIMEOUTS

   超时值

## Appendix B.2. 定义

1. #define ACR_ERROR_PROTO_MISMATCH ((DWORD) 0xA010000FL)

   请求的协议与卡片当前使用的协议不兼容

2. #define ACR_ERROR_UNSUPPORTED_CARD ((DWORD) 0xA0100065L)

   由于 ATR 字符串配置冲突，读写器不能与卡片进行通信。

3. #define ACR_ERROR_UNRESPONSIVE_CARD ((DWORD) 0xA0100066L)

   智能卡没有对复位做出应答。

4. #define ACR_ERROR_UNPOWERED_CARD ((DWORD) 0xA0100067L)

   智能卡电源被切断，不能继续进行通信。

5. #define ACR_ERROR_REMOVED_CARD ((DWORD) 0xA0100069L)

   智能卡已经被移出，不能继续进行通信。

6. #define ACR_PROTOCOL_UNDEFINED 0X00000000

   协议尚未定义。

7. #define ACR_PROTOCOL_T0 0X00000001

   T=0 协议

8. #define ACR_PROTOCOL_T1 0X00000002

   T=1 协议

9. #define ACR_ICC_UNKNOWN 0

   当前处于未知状态。

10. #define ACR_ICC_ABSENT 1

    读写器中没有卡片。

11. #define ACR_ICC_PRESENT 2

    读写器中有卡片

12. #define ACR_ICC_POWERED 3

    正在向卡片供电。

13. #define ICOTL_ACR_CCID_ESCAPE 3500

    用于发送 CCID 直接命令的控制码

14. #define ACR_Open ACR_OpenA

    如果定义了 UNICODE，ACR_Open 会被转换为 ACR_OpenW()函数。

## Appendix B.3. Typedefs

| | |
|---|---|
| 1. typedef struct _ACR_TIMEOUTS | ACR_TIMEOUTS<br>为_ACR_TIMEOUTS 创建一个类型名。 |
| 2. typedef struct _ACR_TIMEOUTS* | ACR_TIMEOUTS<br>为指向_ACR_TIMEOUTS 数据结构的指针创建一个类型名。 |