msdk 接口库

说明书

1. msdk 概述

msdk 接口库支持单头盒子以及双头盒子。

2018 年 1 月以后发货的单双头盒子都是蓝色外壳。和上图中的银白色外壳相比，只是外壳不同，其他功能、性能都一样。如下图所示：

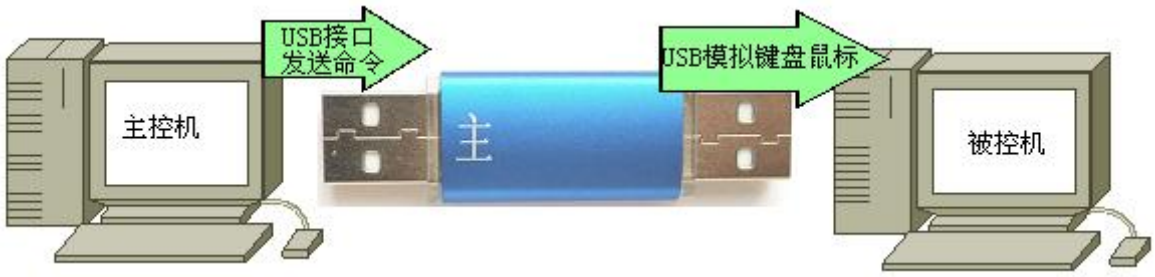
msdk 接口库 msdk.dll 分 X86 的 32 位、X64 的 64 位两个版本。

如果有开发经验，可以使用 X64 的 64 位 msdk.dll，APP 只能编译成 X64 模式，只能在 64 位 windows 上执行。64 位 dll 的功能不全！请谨慎选择！

我们推荐使用 X86 的 32 位 msdk.dll，APP 只能编译成 x86 模式，APP 可以在 32 位、64 位 windows 上执行。

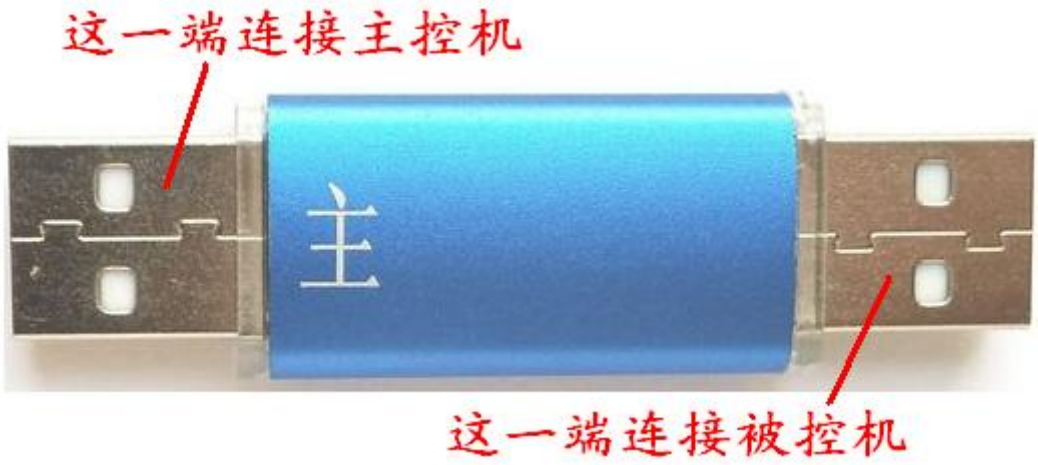
2. 双头盒子简介

盒子可以同时模拟 USB 键盘和鼠标，将从主控机的 USB 接收来的数据转换为键盘或鼠标格式的数据发给被控机，主控机从而可以控制被控机或在被控机的记事本中将数据显示出来。



使用 USB 延长线，将双头盒子连接到主控机和被控机上。主控机和被控机可以是同一台 PC。

可以将双头盒子的被控端映射入虚拟机中，实现主机控制虚拟机。



2.1. 盒子功能、性能

- 全速或低速 USB HID 无驱接口，无需安装驱动、应用软件，即插即用。
- 主控端支持 windows2000/xp/vista/win7/8/10、Linux、Android
- 新功能：鼠标支持绝对移动！详细请看鼠标接口说明！（老盒子不支持）
- 被控机可以是 WII、PS2 游戏机、虚拟机、安卓平板、PC 电脑等只要是能支持实体键盘鼠标的设备。
  - （部分游戏机只支持低速；全速不一定能支持）
- 键盘支持全键盘、组合键操作。可以支持 6+8 个无冲突按键！
  - 8 个控制按键：左/右 Ctrl、左/右 Shift、左/右 Alt、左/右 Windows
  - 6 个除了以上 8 个控制按键外的普通按键
  - 暂不支持多媒体按键（如音量控制按键等）
- 全速盒子：支持每秒 300 组命令（鼠标或/和按键命令）的输入速度

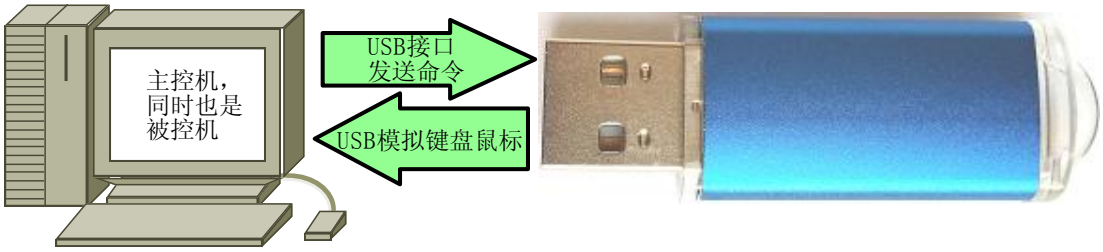
- ◆ 因为低速盒子已经满足应用，全速盒子暂不生产！
- 低速盒子：支持每秒 100 组命令（鼠标或/和按键命令）的输入速度
- 提供 VC/VB/VB.NET/C#/Delphi/Python/JAVA/易语言/按键精灵/TC/Autoit 语言例程
- 每个盒子有唯一序列号
- 支持修改 VID、PID
  - 新盒子支持绝对移动功能可配置：可以配置成有或无绝对移动功能。

3. 单头盒子简介

本盒子只使用一个 USB 接口，模拟 USB 键盘和鼠标，同时可以接收主控机的命令。一台电脑需要同时担当主控机和被控机。盒子从主控机接收数据，转换为键盘或鼠标格式的数据发给被控机。

单头盒子使用时，主控机和被控机只能是同一台 PC。

盒子可以在虚拟机中使用。



3.1. 盒子功能、性能

- 全速 USB HID 无驱接口，无需安装驱动、应用软件，即插即用。
- 主控端支持 windows2000/xp/vista/win7/8/10、Linux
- 键盘支持全键盘、组合键操作。可以支持 6+8 个无冲突按键！
  - 8 个控制按键：左/右 Ctrl、左/右 Shift、左/右 Alt、左/右 Windows
  - 6 个除了以上 8 个控制按键外的普通按键
  - 暂不支持多媒体按键（如音量控制按键等）
- 支持每秒 300 组命令（鼠标或/和按键命令）的输入速度
- 提供 VC/VB/VB.NET/C#/Delphi/Python/JAVA/易语言/按键精灵/TC/Autoit 语言例程
- 每个盒子有唯一序列号
- 支持修改 VID、PID

4. 重点说明

- 盒子不能保存脚本。脚本必须在电脑上运行。可以在按键精灵里编写脚

本或者用 VC/VB/VB.NET/C#/Delphi/Python/JAVA/易语言/按键精灵/TC/Autoit 语言编程，脚本还在电脑上运行，给盒子发命令，盒子就能模拟出相应的按键或鼠标的动作。

- 盒子是硬件模拟键盘鼠标，就和实际的键盘鼠标一样，不能支持后台窗口。

5. 接口说明

以下接口中的 m_hdl 都是指 M_Open 的返回值。

5.1. 通用接口说明

接口函数名	使用说明
M_Open	**打开默认 VID、PID 的端口获取句柄，脚本应用程序启动后只需打开一次端口就可以** HANDLE WINAPI M_Open(int Nbr)
	Nbr 是端口号，无论是双头盒子还是单头盒子，都是从 1 开始，依次为 2/3/4...，最大 126 一台电脑只插入一个盒子，则端口号始终是 1；插入 n 个盒子，端口分别是 1/2/3/4...n 端口和盒子的对应关系，请看“主控机接多个盒子时，打开端口和盒子的对应关系”。 **该接口只能打开使用默认 VID、PID 的盒子。** **对于支持修改 VID、PID 的盒子，如果修改了 VID、PID，则必须使用下述的 M_Open_VidPid 打开。**
	返回非 INVALID_HANDLE_VALUE（该值为-1）即成功
M_Open_VidPid	**打开指定 VID、PID 的单头盒子或者双头盒子的主控端获取句柄** HANDLE WINAPI M_Open(int Vid, int Pid)
	Vid 是厂商标识；Pid 是产品标识 单头盒子的默认 VID PID 是(0xC216, 0x0301) 双头盒子主控端的默认 VID PID 是(0xC216, 0x0102) 0x 是表示十六进制。某些语言（如按键精灵、易语言）不支持无符号的十六进制数，请先将十六进制数转成十进制数后再调用该接口。 使用该接口，一台电脑只能打开一个盒子。如果一台电脑上插入 2 个或多个同样 VID、PID 的盒子，该接口只能打开一个盒子。 所以，**如果一台电脑要插入多个盒子，最好使用支持修改 VID、PID 的盒子，并且将盒子改成不同的 VID、PID**。具体方法可以参考“支持修改 VID、PID 的盒子，可以修改 PID 通过 PID 区分盒子”。
	返回非 INVALID_HANDLE_VALUE（该值为-1）即成功
以上两种打开方式，在一个脚本应用程序里，建议只使用其中一种。 如果**盒子已经修改了 VID、PID，则只能使用 M_Open_VidPid**	

M_Close	**关闭端口；在脚本应用程序退出前再关闭端口** int WINAPI M_Close(HANDLE m_hdl) 返回 0: 成功；!0: 失败
M_GetDevSn	**获取设备序列号** M_GetDevSn(HANDLE m_hdl, DWORD *dwp_LenResponse, unsigned char *ucp_Response)
	参数说明： dwp_LenResponse: 设备序列号的长度，取值范围 0~256。(单位: 字节) ucp_Response: 设备序列号 buf(buf 由调用该 API 的脚本应用程序分配)
	返回值说明： 0 成功 非 0 失败
M_SetUserData	**写用户数据** **该接口仅支持可修改 VID/PID 的单双头盒子。** **使用该接口可以将脚本运行密钥等信息写入盒子中，这些信息将被加密压缩后存入盒子中，这些信息不可读取，只能验证。** M_SetUserData(HANDLE m_hdl, DWORD dw_LenUserData, unsigned char *ucp_UserData)
	参数说明： dw_LenUserData: 数据长度(单位: 字节)，不能超过 256 字节 ucp_UserData: 数据
	返回值说明： 0 成功 非 0 失败
M_VerifyUserData	**验证用户数据** M_VerifyUserData (HANDLE m_hdl, DWORD dw_LenUserData, unsigned char *ucp_UserData)
	参数说明： dw_LenUserData: 数据长度(单位: 字节)，不能超过 256 字节 ucp_UserData: 数据
	返回值说明： 0 成功 非 0 失败
使用以上 M_SetUserData、M_VerifyUserData 两个接口，可以**限制脚本只能使用在某些指定盒子上**：脚本作者在将盒子发给你的客户之前，可以使用 M_SetUserData 先写入一段数据；在脚本里使用 M_VerifyUserData 验证这段数据，只有验证通过，才可以继续使用脚本。	
M_ChkSupportMdy	**检查盒子是否是可修改盒子** int WINAPI M_ChkSupportMdy(HANDLE m_hdl); 返回: 0: 可修改 其他: 不可修改

M_SetNewVidPid	**设置新 VID/PID;** **只支持可修改的单头、双头。普通单头、双头不支持** int WINAPI M_SetNewVidPid(HANDLE m_hdl, int mVid, int mPid, int sVid, int sPid);
	参数说明： mVid: 主控端 Vid; 不能是 C216 或 C217 或 FFFF; 如果是 0，表示 mVid、mPid 不需要更改; mPid 的值将被忽略。 单头主控端 Vid 尾数必须是 0 3 6 9 C F 中其中一个数，如 C300 或 C30C 双头主控端 Vid 尾数必须是 1 4 7 A D 中其中一个数，如 C301 或 C30A mPid: 主控端 Pid; 如果 mVid=0，该参数将被忽略; 不能是 0000 或 FFFF sVid: 被控端 Vid; 如果是单头，该参数将被忽略. 不能是 C216 或 C217 或 FFFF; 如果是 0，表示 sVid、sPid 不需要更改; sPid 的值将被忽略。 双头被控端 Vid 尾数必须是 2 5 8 B E 中其中一个数，如 C302 或 C30B sVid: 被控端 Pid; 如果是单头，该参数将被忽略. 如果 sVid=0，该参数将被忽略; 不能是 0000 或 FFFF
	返回值说明： 0: 成功; -2：该盒子不支持修改; -10: mVID 不符合规则 -11: mPID 不符合规则 -20: sVID 不符合规则 -21: sPID 不符合规则 其他: 修改失败
M_ResetVidPid	**复位盒子的 VID/PID，恢复成出厂设置** **只支持可修改的单头、双头。普通单头、双头不支持** int WINAPI M_ ResetVidPid (HANDLE m_hdl);
M_Delay	**延时指定时间** time:单位 ms int WINAPI M_M_Delay(int time); 该接口实际上是直接调用系统的 Sleep() 脚本应用程序可以不用该接口，直接用系统的 Sleep()

M_DelayRandom	**在指定的最小值和最大值之间延时随机时间** int WINAPI M_DelayRandom(int Min_time, int Max_time); Min_time:最小延时时间; Max_time: 最大延时时间 （单位：ms） 该接口实际上是直接调用系统的 Sleep() 脚本应用程序可以不用该接口，直接用系统的 Sleep()
M_RandDomNbr	**在最小值、最大值之间取随机数** int WINAPI M_RandDomNbr(int Min_V, int Max_V); 该接口实际上是直接调用系统的 rand() 脚本应用程序可以不用该接口，直接用系统的 rand()
以下两个接口将影响键盘鼠标接口的一些配置参数，主要是延时参数；返回值: 0 = 成功	
M_InitParam	**DLL 内部参数恢复默认值** int WINAPI M_InitParam(HANDLE m_hdl); 该接口将 DLL 内部的参数恢复成默认值。 该接口将影响以下几个接口： 键盘接口： M_KeyPress 、 M_KeyPress2 、 M_KeyInputString 、 M_KeyInputStringGBK、M_KeyInputStringUnicode 鼠标接口： M_LeftClick 、 M_LeftDoubleClick 、 M_RightClick 、 M_MiddleClick、M_MoveR、M_MoveTo、 M_MoveR2、M_MoveTo2、 M_MoveTo3
M_SetParam	**设置 DLL 内部参数** int WINAPI M_SetParam (HANDLE m_hdl, int ParamType, int Param1, int Param2); 该接口可以调整 DLL 内部的配置参数。参数说明请看下表。 注意：这些参数不保存在盒子里，也不保存在 DLL 里。在程序打开端口后，对应盒子的参数将立即恢复成默认值。如果需要调整参数，程序需要在打开端口后，调用该接口设置参数，然后再操作键盘鼠标接口。设置参数后，参数立即生效！

Param Type	Param1	Param2	说明	影响到的接口	注意
0	最小值	最大值	单击按键，按下和弹起之间的延时值（默认值是 50,80）	M_KeyPress M_KeyPress2 M_KeyInputString M_KeyInputString GBK M_KeyInputString Unicode	实际延时时间在最小最大值之间随机
2	最小值	最大值	多个按键，每个按键之间的延时值（默认值是 150,600）		
8	最小值	最大值	单击鼠标(左中右键)，按下和弹起之间的延时值（默认值是 50,80）	M_LeftClick M_RightClick M_MiddleClick	
10	最小值	最大值	多次单击鼠标(左中右键)，每次单击之间的延时值（默认值是 500,900）		
12	最小值	最大值	双击鼠标(左键)，两次单击之间的延时值（默认值是 60,110）	M_LeftDoubleClick	
14	最小值	最大值	多次双击鼠标(左键)，每次双击之间的延时值（默认值是 500,900）		
20	最小值	最大值	鼠标移动轨迹由多条直线组成，每条直线移动之间的延时值（默认值是 10,20）	M_MoveR M_MoveTo M_MoveR2 M_MoveTo2 M_MoveTo3	

5.2. 键盘接口说明

以下的接口函数，返回值： 0 = 成功；-1 = 失败

接口函数名	使用说明
注意：以下 4 个接口中的 HidKeyCode 是 USB HID 的键盘码，不是 Windows 的标准键盘码。键盘码详见附录 1。	

M_KeyPress	**单击(按下后立刻弹起)指定按键** int WINAPI M_KeyPress(HANDLE m_hdl, int HidKeyCode, int Nbr); HidKeyCode: 键盘码（详见附录 1）。 Nbr: 按下次数 注意 1：在 DLL 内部，M_KeyPress 是由 M_KeyDown + 50~80ms 的随机延时 + M_KeyUp 组成。 注意 2：在 DLL 内部，两次 M_KeyPress 单击之间的延时是 150~600ms 随机延时。 注意 3：如果需要自己控制按下和弹起之间的延时，可以在脚本应用程序里 M_KeyDown + 自定义延时 + M_KeyUp。也可以调用 M_SetParam 设置
M_KeyDown	**按下指定按键不弹起，如果按下不弹起，可以和其他按键组成组合键** int WINAPI M_KeyDown(HANDLE m_hdl, int HidKeyCode); HidKeyCode: 键盘码（详见附录 1）。
M_KeyUp	**弹起指定按键** int WINAPI M_KeyUp(HANDLE m_hdl, int HidKeyCode); HidKeyCode: 键盘码（详见附录 1）。
M_KeyState	**读取按键状态；返回值：0=弹起状态；1:=按下状态** 使用该接口，不允许手工操作键盘，否则该接口返回值有可能不正确 int WINAPI M_KeyState(HANDLE m_hdl, int HidKeyCode); HidKeyCode: 键盘码（详见附录 1）。
注意：以下 4 个接口中的 KeyCode 是 Windows 的标准键盘码（可参考附录 2 的 VK 键值），易语言和按键精灵可以直接使用它们自带的按键码值（如：A 键，按键码值是 65）。	
M_KeyPress2	**单击(按下后立刻弹起)指定按键** int WINAPI M_KeyPress2(HANDLE m_hdl, int KeyCode, int Nbr); KeyCode: Windows 标准键盘码。 Nbr: 按下次数 注意 1：在 DLL 内部，M_KeyPress2 是由 M_KeyDown2 + 50~80ms 的随机延时 + M_KeyUp2 组成。 注意 2：在 DLL 内部，两次 M_KeyPress2 单击之间的延时是 150~600ms 随机延时。 注意 3：如果需要自己控制按下和弹起之间的延时，可以在脚本应用程序里 M_ KeyDown2 + 自定义延时 + M_ KeyUp2。也可以调用 M_SetParam 设置
M_KeyDown2	**按下指定按键不弹起，如果按下不弹起，可以和其他按键组成组合键** int WINAPI M_KeyDown2(HANDLE m_hdl, int KeyCode); KeyCode: Windows 标准键盘码。

M_KeyUp2	**弹起指定按键** int WINAPI M_KeyUp2(HANDLE m_hdl, int KeyCode); KeyCode: Windows 标准键盘码。
M_KeyState2	**读取按键状态；返回值：0=弹起状态；1:=按下状态** 使用该接口，不允许手工操作键盘，否则该接口返回值有可能不正确 int WINAPI M_KeyState2(HANDLE m_hdl, int KeyCode); KeyCode: Windows 标准键盘码。
M_ReleaseAllKey	**弹起所有按键。如果出现按键异常，也可以调用该接口恢复。** int WINAPI M_ReleaseAllKey(HANDLE m_hdl);
M_NumLockLedState	**读取小键盘 NumLock 灯的状态；返回值：0:灭；1:亮；-1：失败** int WINAPI M_NumLockLedState(HANDLE m_hdl)
M_CapsLockLedState	**读取 CapsLock 灯的状态；返回值：0:灭；1:亮；-1：失败** int WINAPI M_CapsLockLedState(HANDLE m_hdl)
M_ScrollLockLedState	**读取 ScrollLock 灯的状态；返回值：0:灭；1:亮；-1：失败** int WINAPI M_ScrollLockLedState(HANDLE m_hdl)
M_KeyInputString	**输入一串 ASCII 字符串，如"ABCdef012,.<>"** 使用该接口，不允许手工操作键盘，否则字符有可能不能正确输入 int WINAPI M_KeyInputString(HANDLE m_hdl, char *InputStr, int InputLen) InputStr: 输入缓冲区首地址 InputLen: 字符长度 **注意 1：不支持\n\r 等转义字符！不支持中文！** **注意 2：如果在 InputLen 个字节内有非 ASCII 字符，这些字符将会被忽略。** **注意 3：某些语言只能用 unicode 编码的字符串（如按键精灵），需要注意将 InputLen 的长度设置为实际 ASCII 字符数的两倍。** **注意 4：调用此接口前，建议先释放所有按键，避免可能存在的组合键问题导致无法正确输入。比如，如果在调用此接口前，已经按下 Ctrl 键还未释放，那调用该接口将无法正常输入字符。**
以下两个中文输入接口不一定适用于所有输入框。请先手工测试确认，方法如下：将光标点到输入框内，按下 Alt 键，小键盘依次输入 45217，弹起 Alt，如果能出现“啊”，则表示可以使用这两个接口输入中文。 **以下两个接口，请根据编程语言所用的字符串存储方式进行选用。如果不清楚是哪一种，请两种都尝试。**	

M_KeyInputStringGBK	**输入一串字符串，支持中文(GBK 编码)英文混合，如"啊啊啊ABCdef012,.<>"** 使用该接口，不允许手工操作键盘，否则字符有可能不能正确输入 int WINAPI M_ KeyInputStringGBK(HANDLE m_hdl, char *InputStr, int InputLen) InputStr: 输入缓冲区首地址 InputLen: 字符长度（注意，每个中文（包括中文符号）占 2 个字节，英文 ASCII 每个占 1 字节。），最大不能超过 512 字节 **注意 1：该接口只能支持 GBK 编码的中文。如果在 InputLen 个字节内有非 GBK 编码的中文或者非 ASCII 字符，这些字符将会被忽略或者会出现乱码。** **注意 2：不支持\n\r 等转义字符！** **注意 3：调用此接口前，建议先释放所有按键，避免可能存在的组合键问题导致无法正确输入。比如，如果在调用此接口前，已经按下 Ctrl 键还未释放，那调用该接口将无法正常输入字符。**
M_KeyInputStringUnicode	**输入一串字符串，支持中文(Unicode 编码)英文混合，如"啊啊啊ABCdef012,.<>"** 使用该接口，不允许手工操作键盘，否则字符有可能不能正确输入 int WINAPI M_KeyInputStringUnicode (HANDLE m_hdl, char *InputStr, int InputLen) InputStr: 输入缓冲区首地址 InputLen: 字符长度(注意，无论中文(包括中文符号)、英文 ASCII，每个占 2 个字节)，最大不能超过 512 字节 **注意 1：该接口只能支持 Unicode 编码的中文。如果在 InputLen 个字节内有非 Unicode 编码的中文或者非 ASCII 字符，这些字符将会被忽略或者会出现乱码。** **注意 2：不支持\n\r 等转义字符！** **注意 3：调用此接口前，建议先释放所有按键，避免可能存在的组合键问题导致无法正确输入。比如，如果在调用此接口前，已经按下 Ctrl 键还未释放，那调用该接口将无法正常输入字符。**

5.3. 鼠标接口说明

以下的接口函数，返回值： 0 = 成功；-1 = 失败

接口函数名	使用说明

M_LeftClick	**左键单击(按下后立刻弹起)** int WINAPI M_LeftClick(HANDLE m_hdl, int Nbr); Nbr: 单击次数 注意 1：在 DLL 内部，M_LeftClick 是由 M_LeftDown + 50~80ms 的随机延时 + M_LeftUp 组成。 如果需要自己控制按下和弹起之间的延时，可以在脚本应用程序里 M_LeftDown + 自定义延时 + M_LeftUp。 注意 2：在 DLL 内部，两次 M_LeftClick 之间的延时是 500~900ms 随机延时。
M_LeftDoubleClick	**左键双击** int WINAPI M_LeftDoubleClick(HANDLE m_hdl, int Nbr); Nbr: 双击次数 注意 1：在 DLL 内部，M_LeftDoubleClick 是由 M_LeftClick + 60~110ms 的随机延时 + M_LeftClick 组成。 注意 2：在 DLL 内部，两次 M_ LeftDoubleClick 之间的延时是 500~900ms 随机延时。
M_LeftDown	**按下左键不弹起** int WINAPI M_LeftDown(HANDLE m_hdl);
M_LeftUp	**弹起左键** int WINAPI M_LeftUp(HANDLE m_hdl);
M_RightClick	**右键单击(按下后立刻弹起)** int WINAPI M_RightClick(HANDLE m_hdl, int Nbr); Nbr: 单击次数 注意 1：在 DLL 内部，M_RightClick 是由 M_RightDown + 50~80ms 的随机延时 + M_RightUp 组成。 如果需要自己控制按下和弹起之间的延时，可以在脚本应用程序里 M_RightDown + 自定义延时 + M_RightUp。 注意 2：在 DLL 内部，两次 M_RightClick 之间的延时是 500~900ms 随机延时。
M_RightDown	**按下右键不弹起** int WINAPI M_RightDown(HANDLE m_hdl);
M_RightUp	**弹起右键** int WINAPI M_RightUp(HANDLE m_hdl);

M_MiddleClick	**中键单击(按下后立刻弹起)** int WINAPI M_MiddleClick(HANDLE m_hdl, int Nbr); Nbr: 单击次数 注意 1：在 DLL 内部，M_MiddleClick 是由 M_MiddleDown + 50~80ms 的随机延时 + M_MiddleUp 组成。 如果需要自己控制按下和弹起之间的延时，可以在脚本应用程序里 M_MiddleDown + 自定义延时 + M_MiddleUp。 注意 2:在 DLL 内部,两次 M_MiddleClick 之间的延时是 500~900ms 随机延时。
M_MiddleDown	**按下中键不弹起** int WINAPI M_MiddleDown(HANDLE m_hdl);
M_MiddleUp	**弹起中键** int WINAPI M_MiddleUp(HANDLE m_hdl);
M_ReleaseAllMouse	**弹起鼠标的所有按键（包括左键、中键、右键）** int WINAPI M_ReleaseAllMouse(HANDLE m_hdl);
M_MouseKeyState	**读取鼠标左中右键状态** int WINAPI M_MouseKeyState(HANDLE m_hdl, int MouseKeyCode) MouseKeyCode: 1=左键 2=右键 3=中键 返回 0: 弹起状态；1:按下状态；-1: 失败 **注意：只能读取盒子中鼠标的状态，读取不到实体鼠标的状态**
M_MouseWheel	**滚动鼠标滚轮** int WINAPI M_MouseWheel(HANDLE m_hdl,int Nbr); Nbr: 滚动量， 为正,向上滚动；为负，向下滚动;
M_ResetMousePos	**将鼠标移动到原点(0,0)** int WINAPI M_ResetMousePos(HANDLE m_hdl); 在出现移动出现异常时，可以用该函数将鼠标复位
M_MoveR	**从当前位置相对移动鼠标** int WINAPI M_MoveR(HANDLE m_hdl,int x, int y); x: x 方向（横轴）的距离（正:向右；负值:向左）; y: y 方向（纵轴）的距离（正:向下；负值:向上） **如果只是本机使用，请使用下面的 M_MoveR2**

M_MoveTo	**移动鼠标到指定坐标** **使用该接口请注意：该函数只适合完全由脚本应用程序控制鼠标移动的应用方式。！！** int WINAPI M_MoveTo(HANDLE m_hdl,int x, int y); x: x 方向（横轴）的坐标; y: y 方向（纵轴）的坐标。 注意：坐标原点(0, 0)在屏幕左上角 **注意：如果出现过将鼠标移动的距离超过屏幕大小，再次 M_MoveTo 可能会出现无法正确移动到指定坐标的问题！如果出现该问题，需调用 M_ResetMousePos 复位** **这个接口在第一次调用时，会先移动到原点。以后再调用这个接口，dll 会记住每次调用的距离，然后计算需要相对移动的距离。如果在脚本应用程序运行中，有手工操作鼠标，那 dll 记住的坐标就不准确了。再调用这个接口移动鼠标，移动的坐标也就不准确。** **1、如果只是本机使用，请使用下面的 M_MoveTo2** **2、强烈建议新写脚本使用 M_MoveTo3 接口，不会再有以上困扰。** **3、鼠标移动是模拟曲线运动。移动方法请看下述的“鼠标几个移动接口比较”**
M_GetCurrMousePos	**读取当前鼠标所在坐标** **使用该接口请注意：该函数只适合完全由脚本应用程序控制鼠标移动的应用方式。！！** int WINAPI M_GetCurrMousePos(HANDLE m_hdl,int *x, int *y); 返回的坐标值在 x、y 中。 **注意：该函数不是通过调用 windows 的 API 接口来获取被控机的鼠标当前坐标，而是通过监测脚本应用程序调用 M_MoveTo/M_MoveTo3 来实时调整和记录当前坐标值，所以如果在调用 M_MoveTo/M_MoveTo3 之外，还手工调整被控机的鼠标，那调用该函数读取到的坐标值将不准确。** **注意：该函数必须在执行一次 M_MoveTo/M_MoveTo3 或 M_ResetMousePos 函数后才能正确执行！** **注意：如果曾经出现过将鼠标移动的距离超过屏幕大小，这里读取到的坐标值有可能是不正确的！如果出现该问题，需调用 M_ResetMousePos 复位** **如果只是本机使用，请使用下面的 M_GetCurrMousePos2**
M_GetCurrMousePosX	**读取当前鼠标 X 坐标值** int WINAPI M_GetCurrMousePosX(HANDLE m_hdl); 返回值为当前鼠标 X 坐标值。 **注意：该函数功能和上述 M_GetCurrMousePos 的功能重复，只是为了方便某些不支持指针的语言使用，如按键精灵。**
M_GetCurrMousePosY	**读取当前鼠标 Y 坐标值** int WINAPI M_GetCurrMousePosY(HANDLE m_hdl); 返回值为当前鼠标 Y 坐标值。 **注意：该函数功能和上述 M_GetCurrMousePos 的功能重复，只是为了方便某些不支持指针的语言使用，如按键精灵。**

注意，以下接口仅适用主控机和被控机是同一台电脑的使用方式(单头盒子；双头盒子的两个 USB 头都连接到同一台电脑) **以下接口 DLL 将调用 windows 操作系统的 API 来获取当前鼠标位置，计算坐标偏差后，通过相对移动方式移动到目的坐标。DLL 将不记录鼠标移动的位置。**	
M_MoveR2	**从当前位置相对移动鼠标 2** int WINAPI M_MoveR2(HANDLE m_hdl,int x, int y); x: x 方向（横轴）的距离（正:向右；负值:向左）； y: y 方向（纵轴）的距离（正:向下；负值:向上）
M_MoveTo2	**移动鼠标到指定坐标 2** int WINAPI M_MoveTo2(HANDLE m_hdl,int x, int y); x: x 方向（横轴）的坐标; y: y 方向（纵轴）的坐标。 注意：坐标原点(0, 0)在屏幕左上角 **鼠标移动是模拟曲线运动。移动方法请看下述的“鼠标几个移动接口比较”**
M_GetCurrMousePos2	**读取当前鼠标所在坐标** int WINAPI M_GetCurrMousePos(int *x, int *y); 返回的坐标值在 x、y 中。 该接口实际上是直接调用系统的 GetCursorPos () 脚本应用程序可以不用该接口，直接用系统的 GetCursorPos ()
注意，以下接口将使用绝对移动功能。也就是说，移动到指定坐标将不再需要移动到原点，不需要在系统设置中先设置鼠标，每次移动的精度也不会受到手工移动实体鼠标的影响。在使用绝对移动前，必须先输入被控机的分辨率。打开端口后，只需要调用一次 M_ResolutionUsed 就可以！ **注意：在某些坐标上，会有±2 个像素的误差。** **注意：老盒子不支持该功能！2017.5 月起购买的盒子支持该功能，具体请淘宝或 QQ 联系！**	
M_ResolutionUsed	**输入被控机的屏幕分辨率** **返回值如果是-10，表示该盒子不支持绝对移动功能。返回 0 表示执行正确。可以用该接口判断盒子是否支持绝对移动功能** int WINAPI M_ResolutionUsed(HANDLE m_hdl,int x, int y); x: x 方向 y: y 方向 比如：屏幕分辨率是 1024×768，那么 x=1024，y=768
M_MoveTo3	**移动鼠标到指定坐标 3** int WINAPI M_MoveTo3(HANDLE m_hdl,int x, int y); x: x 方向的坐标; y: y 方向的坐标。 注意：坐标原点(0, 0)在屏幕左上角 **鼠标移动是模拟曲线运动。移动方法请看下述的“鼠标几个移动接口比较”**

M_MoveTo3	**一步到位移动鼠标到指定坐标 3** int WINAPI M_MoveTo3_D(HANDLE m_hdl,int x, int y); x: x 方向的坐标; y: y 方向的坐标。 注意：坐标原点(0, 0)在屏幕左上角 **鼠标移动是一步到位，没有移动轨迹！**

6. 鼠标的相对移动、绝对移动的说明

2017 年 5 月新推出的盒子新增支持鼠标的绝对移动，下面介绍什么是相对移动，什么是绝对移动。

6.1. 相对移动

我们日常用的普通鼠标都是相对移动。鼠标并不知道也不关心当前电脑屏幕上的光标的具体坐标位置。鼠标被移动后，鼠标将告诉电脑，向左向右向上向下移动了多少，对应的，电脑屏幕上的光标将从**当前位置**向左向右向上向下移动。

我们的单头盒子、双头盒子都有相对移动功能（具体请看鼠标接口说明中的 M_MoveTo/M_MoveR/M_MoveTo2/M_MoveR2）。

既然盒子只有相对移动功能，那为什么会有 M_MoveTo/M_MoveTo2 这两个接口可以移动到绝对坐标位置？

对于 M_MoveTo 接口，脚本应用程序在第一次调用该接口时，DLL 将强制将光标移动到原点，然后再通过相对移动方法将光标移动到绝对坐标位置。DLL 内部会记录鼠标移动的过程，从而能够知道光标的当前位置。但是，如果手工移动了实体鼠标，改变了光标的位置，那再次调用 M_MoveTo 移动鼠标，则移动到的绝对坐标位置将出错。

对于 M_MoveTo2 接口，DLL 将调用 Windows 提供的 API（getcursorpos）获取本机鼠标位置，计算坐标偏差后，通过相对移动方式将光标移动到目的坐标。DLL 不记录鼠标移动的位置。该接口只适用于单头盒子或双头盒子的两个 usb 头都在一个主机里（而且不能一个在本机，一个在虚拟机）。

使用以上接口，还需要在被控机上设置鼠标后，鼠标移动位置才能准确。具体请看鼠标接口说明。

6.2. 绝对移动

绝对移动功能可以近似理解为触摸屏。当我们在触摸屏上操作时，屏幕上的光标移动就是绝对移动。新版盒子提供 M_MoveTo3 接口调用绝对移动功能。被控机不再需要设置鼠标，手工动实体鼠标也不会影响下一次鼠标移动的精度。

注意：2017 年 5 月以后销售的支持修改 VID、PID 的盒子还支持绝对移动功能可配置：可以配置成有或无绝对移动功能！如果用不上绝对移动功能，又不想在设备管理器里多一个鼠标设备，可以将绝对移动功能通过配置方式去掉。

2016 年 11 月到 2017 年 4 月之间发货的盒子也具有绝对移动功能，这些盒子的绝对移动功能有一些问题：绝对移动会有过冲，也就是移动过程中会冲过指定坐标，但是最后还是会移动到指定坐标。另外，如果一直用绝对移动，会出现鼠标左右键单击或滚轮滚动的问题。

如果购买了这些盒子，请谨慎使用绝对移动功能。

注意：2017 年 5 月以后购买的盒子已经没有上述问题！

6.3. 鼠标几个移动接口比较

盒子提供 3 种鼠标移动方法，可以混合调用。

	M_MoveR M_MoveTo M_GetCurrMousePos	M_MoveR2 M_MoveTo2 M_GetCurrMousePos2	M_MoveTo3 M_MoveTo3_D
移动方法	M_MoveTo3_D 是一步到位，除了 M_MoveTo3_D，其他接口的移动方法都和下述描述一样： 在 dll 里，x 或 y 每次只直线移动 100~127 个像素，每次移动之间有 10~20ms 的延时。如果移动距离小于 100 个像素，是直线移动过去；如果大于 100 个像素，移动轨迹是多条直线组成的**模拟曲线**。 如果想缩短移动时间，可以自己在脚本里每次移动不超过 100 个像素，每次移动之间延时时间自己定。对于双头盒子，每秒不要超过 100 次移动；单头盒子没有限制。		
适用于	单双头	单头； 如果是双头，两个 usb 头必须都在一个主机里，同时不能一个在本机，一个在虚拟机	单双头
如果被控机的屏幕分辨率不可知	可以用	可以用	不可以用
能否调用 M_ResetMousePos	可以调用	可以调用	不可以调用 可以用 M_MoveTo3(0,0) 代替
被控机鼠标是否需要设置	需要。如果不设置，移动不准确	需要。如果不设置，移动不准确	不需要。 但需要输入被控机分辨率
鼠标移动精度	设置后，精度准确	设置后，精度准确	在某些坐标上会有最大 ±2 的误差
支持的系统	Windows/Linux/安卓	Windows/Linux/安卓	Windows/Linux
盒子技术特点	所有移动都是相对移动	所有移动都是相对移动	所有移动都是绝对移动

DLL 技术特点	第一次调用 M_MoveTo 将强制回到原点，以后每次移动将计算并记录移动过程。	调用 Windows 提供的 API 获取本机鼠标位置，计算坐标偏差后，通过相对移动方式移动到目的坐标。DLL 将不记录鼠标移动的位置。	DLL 内部只记录最后一次鼠标移动的位置，以便脚本应用程序读取。
手工移动实体鼠标	会影响下次移动的精度。	不会影响下次移动的精度。不会影响鼠标移动模拟的曲线	不会影响下次移动的精度。 但是，为了模拟曲线移动，鼠标移动的起始点不是从当前实际鼠标位置，而是从上一次 DLL 记录的鼠标位置开始移动。

7. 关于盒子端口打开失败

1. 先检查盒子是否已经被电脑识别：
   a) 请将设备管理器打开。（右键点“我的电脑”，选设备管理器，或者选管理->设备管理器）
   b) 请将盒子插入电脑 USB 口。双头盒子请将主控端插入电脑。
   c) 设备管理器应该会闪一下，如果没有闪，则有可能是盒子问题。
   d) 如果设备管理器闪一下或几下，并且没有新增未识别设备（打感叹号的设备），请继续下面的检查。
2. 请检查盒子是否已经被修改过 VID、PID，如果已经修改，请使用 M_Open_VidPid 接口打开盒子端口。如果不记得修改后盒子的 VID、PID 请查看“修改 VID、PID 后，如何查到当前盒子的 VID、PID”。
3. 经过以上检查，仍不能用“快速测试程序”打开端口，则有可能是 HID 服务没有启动。请点击“快速测试程序”的“开启 HID 服务”按钮。重启电脑后，重新尝试打开端口。

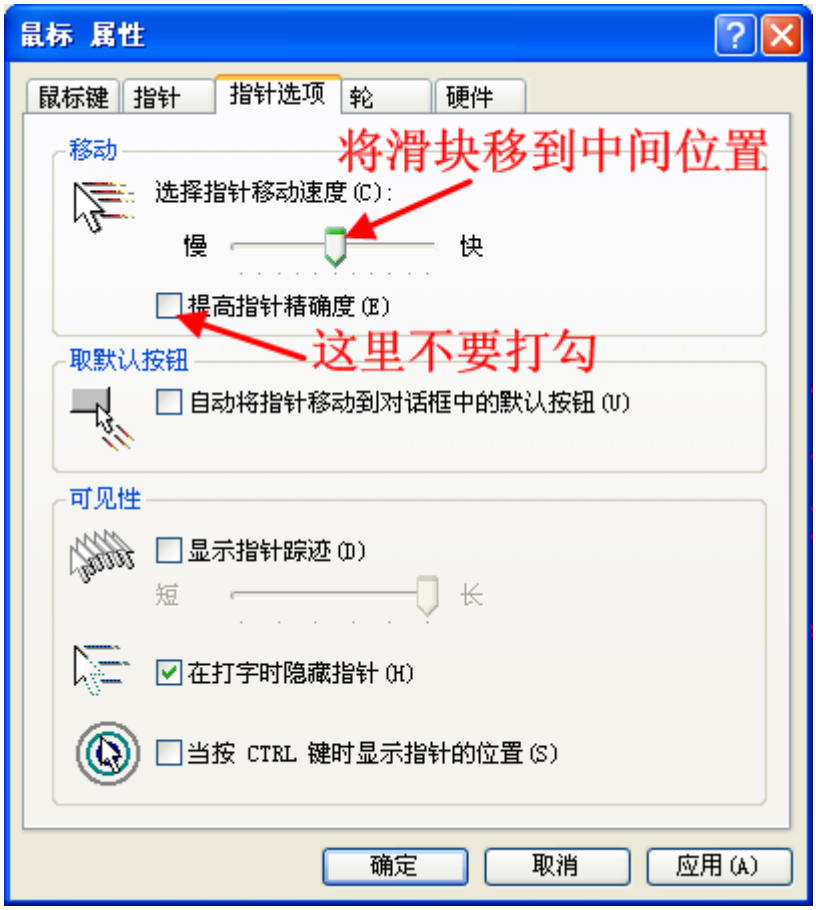
8. 关于鼠标移动位置不准确的说明

注意：如果使用绝对坐标移动功能 M_MoveTo3，则不需要按下述设置。

由于操作系统对鼠标移动进行了优化，导致控制鼠标移动的距离和实际移动距离不相符。需要修改鼠标配置就可以解决这个问题。

以 WinXp 系统为例，修改方法如下：

开始 -> 设置 -> 控制面板 -> 鼠标 -> 指针选项，然后如下图修改：



9. 关于主控机和被控机在同一虚拟机中，鼠标无法控制的问题

如果主控机和被控机都在同一个虚拟机中，有可能无法通过脚本应用程序控制鼠标的动作。如果出现这种问题，以 VirtualBox 为例，请点击 虚拟机 - 控制 - 禁止自动独占鼠标。

10. 主控机接多个盒子时，打开端口和盒子的对应关系

10.1. 不支持修改 VID、PID 的盒子，只能通过端口区分盒子

如果主控机同时接了多个盒子，打开端口号和盒子插入顺序没有必然的对应关系。为了简化操作，建议采用如下方法来使得端口和盒子之间有必然的对应关系：

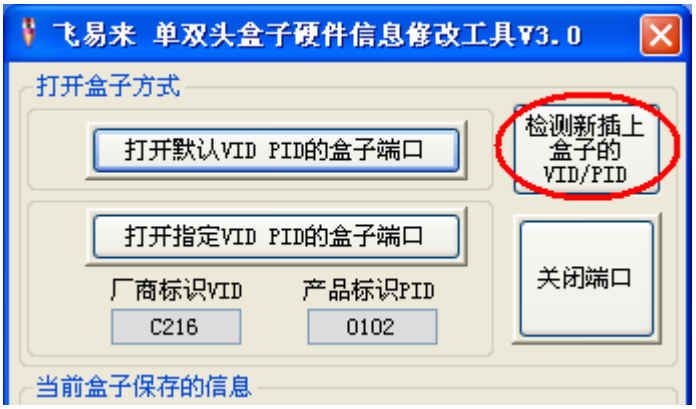
1. 不要打开任何端口，也不要打开任何和盒子相关的脚本应用程序。
2. 将所有需要操作的盒子都插入到主控机上，将所有盒子都连接到被控机上。
   记住插在主控机上的 USB 口位置；记住插入的盒子个数。
3. 在主控机上打开脚本应用程序，打开端口 1，调用 MoveTo 命令移动鼠标，查看是哪个被控机上的鼠标有移动，进而确定对应的盒子，将该盒子所插的 USB 口位置标注为 1。
   打开端口 2，按同样方法操作，标注第 2 个 USB 口位置。
   依次打开所 有端口，标注所有插有盒子的 USB 口位置。
4. 至此，端口和盒子的对应关系已经确定。今后操作时，只要插入同样数量的盒子、插在同样的 USB 口位置，在插好所有盒子后，再打开脚本应用程序，从端口 1 开始打开各个端口，则端口 1 打开的就是 USB 口位置标注为 1 的盒子，端口 2 打开的就是 USB 口位置标注为 2 的盒子，以此类推。
   即便盒子可能会更换成另外一批新的盒子，即便电脑重启，这个对应关系也不会改变。
   如果出现操作错误，导致打开端口和盒子对应关系错乱，建议关闭脚本应用程序后再重新打开脚本应用程序；如果这样仍没有解决问题，建议把脚本应用程序关闭，把所有盒子都拔掉，再重新插上，等 10 秒后，再打开脚本应用程序继续操作。

10.2. 支持修改 VID、PID 的盒子，可以修改 PID 通过 PID 区分盒子

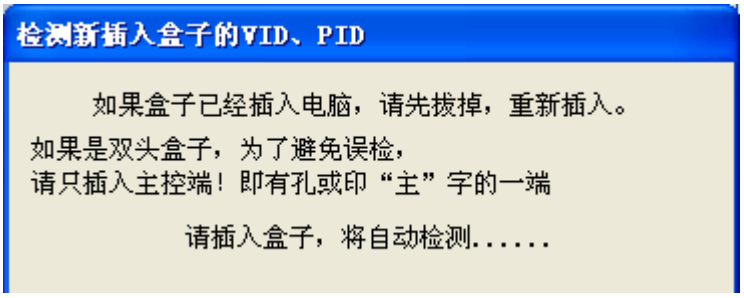
建议将所有盒子的 PID 都修改成不同的，使用 M_Open_VidPid 打开操作，这样就不用考虑上述端口对应的问题。

11. 修改 VID、PID 后，如何查到当前盒子的 VID、PID

1. 打开修改工具，点击右上角的“检测新插上盒子的 VID/PID”按钮



2. 在弹出的框中，按照提示重新插上主控端，



3. 找到 vid pid 后，点击“将 VID PID 输入...”。

4. 如果要恢复成出厂的 VID、PID，在修改工具中，点击

打开端口后，点“恢复出厂设置”按钮，插拔后生效。

12. 新盒子具有设备名，虚拟机映射时可看到该设备名

2017.5.1 之后购买的单双头盒子，在绝对移动功能是打开的状态下，盒子具有设备名。在虚拟机映射时，将可以看到该设备名。

盒子类型	有设备名？
2017.5.1 之前销售的普通单双头盒子	无
2017.5.1 之前销售的支持修改	无

VID/PID 的单双头盒子	
2017.5.1 之后销售的普通单双头盒子	有
2017.5.1 之后销售的支持修改 VID/PID 的单双头盒子	如果绝对移动功能是默认的打开状态：有 如果绝对移动功能是关闭状态：无

设备名定义如下

	设备名规则	默认 PID	默认设备名
单头盒子	DPD+当前盒子的 PID	0301	DPD0301
双头盒子主控端	MPD+当前盒子主控端的 PID	0102	MPD0102
双头盒子被控端	SPD+当前盒子被控端的 PID	0209	SPD0209

对于可修改 VID/PID 的盒子，当一台电脑插入多个盒子时，**可以通过修改 PID，将盒子的设备名修改成不同的名称，这样在虚拟机映射时，就好区分了。**

“DPD”、“MPD”、“SPD”这三个前缀字符目前不可以更改！

13. 将盒子映射到虚拟机中

以下以 VMware 和 VirtualBox(以下简称 Vbox)为例说明如何设置虚拟机。

13.1. VMware 虚拟机

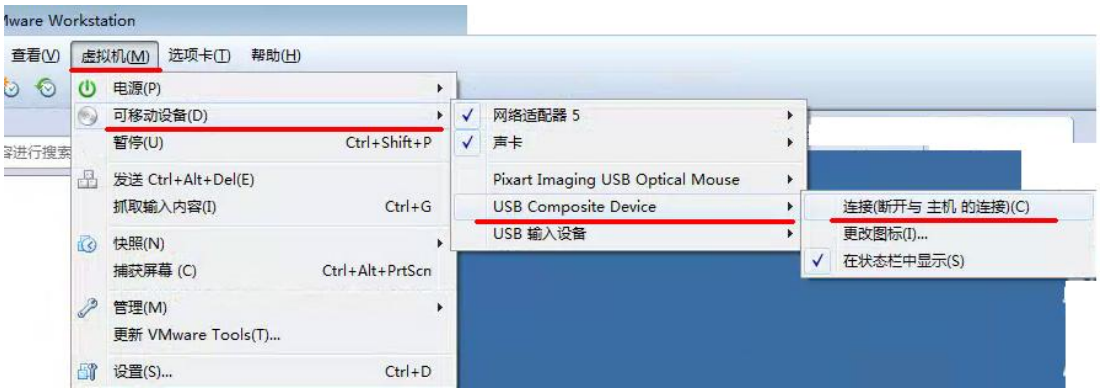
13.1.1. 先设置虚拟机使能 USB

13.1.2. 启动虚拟机，进行映射

插上盒子，如果是双头盒子，请用 USB 延长线将两个 USB 头都连接到电脑。启动虚拟机，然后：

13.1.2.1. 无设备名时的单头盒子、双头盒子被控机

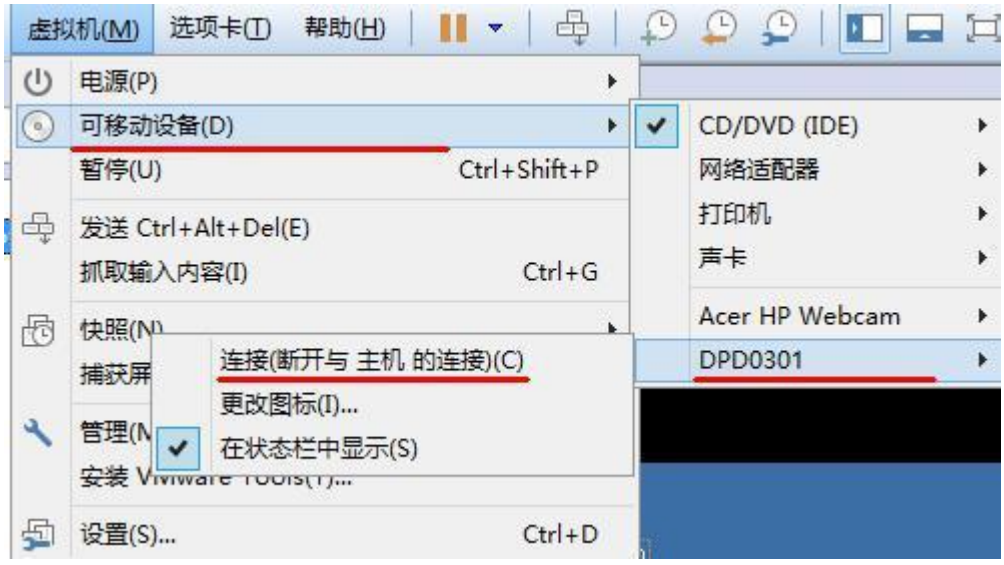
如下图设置映射：虚拟机 -> 可移动设备 -> USB Composite Device -> 连接



映射结束后。可以在虚拟机设备管理器中，查看是否增加了一个鼠标和一个键盘。

13.1.2.2. 有设备名时的单头盒子、双头盒子主控端和被控机

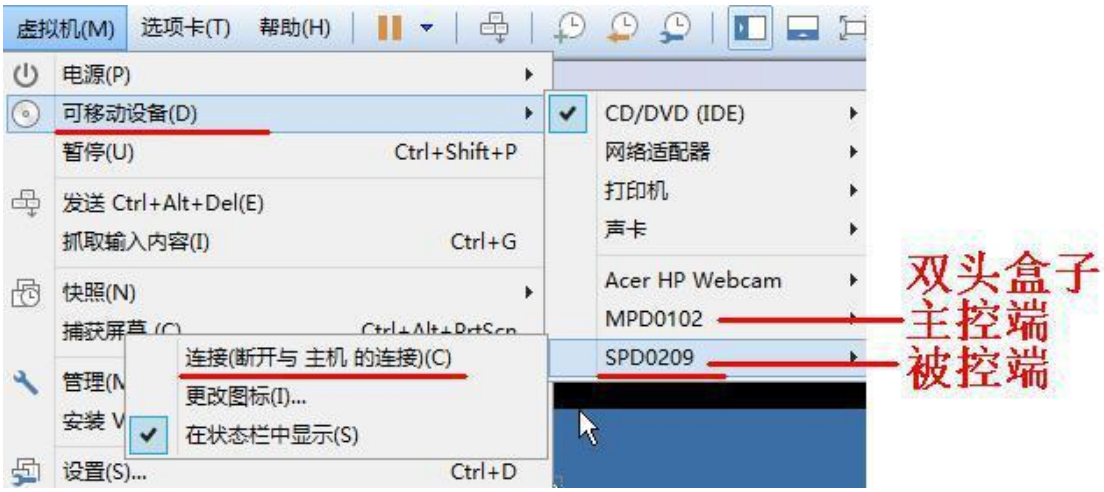
如下图设置映射：虚拟机 -> 可移动设备 -> 设备名 -> 连接



映射结束后。可以在虚拟机设备管理器中，查看是否增加了一个鼠标和一个键盘。

注意：上述 DPD0301 里的 0301 是指当前单头盒子的 PID。如果使用可修改 VID/PID 的单头盒子，并且修改过 PID，则这里显示的是 DPD+修改后的 PID。比如修改后的 PID 是 0001，则这里就是 DPD0001。

下面是双头盒子主控端、被控端的映射方法：



13.1.2.3. 上述操作后的注意事项

- 如果弹出“无法连接到此虚拟机的……”“主机需要输入此设备”的提示框，请找到虚拟机对应的配置文件 vmx 文件（可以搜索*.vmx 找到 vmx 文件），在设备配置文件.vmx 里添加一行

  usb.generic.allowLastHID = "TRUE"
- 如果点击连接后，虚拟机里的设备管理器并没有看到新增设备，请打开虚拟机中的设备管理器，选择操作——扫描检测硬件改动。

13.1.3. 如果虚拟机无法看到 USB COMPOSITE 设备

如果在移动设备中没有找到 USB Composite Device 的设备，也没有其他 USB 设备，如下图：

请确认以下几点：

1、你的虚拟机是否是买来的？如果是买来的，而你又必须要用这个虚拟机，请联系卖主，让他把 USB 组件加进去。**对于这种虚拟机，请自行找虚拟机的卖家解决映射问题，我们不提供支持！**

如果不是必须要用买来的虚拟机，建议下载安装包，重新从头安装一个。

2、盒子所插的 USB 口是否是 USB3.0？USB3.0 口里面一般是蓝色的，USB2.0 口里面一般是黑色或者白色。如下图：

一些虚拟机不能支持 USB3.0 口，请将盒子插到 USB2.0 口上再尝试映射。

13.1.4. 关于 VMware 虚拟机的自动映射

VMware 虚拟机在第一次手工映射后，下次重新启动虚拟机时：

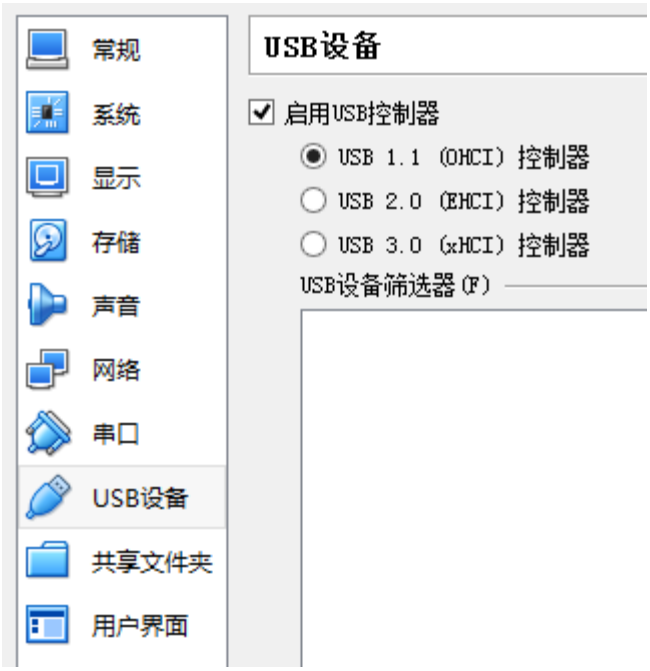
1. 单头盒子可以自动映射进虚拟机，不用再手工映射。非常适合批量。
2. 双头盒子主控端可以自动映射进虚拟机。
3. 双头盒子被控端不能自动映射进虚拟机。（可以通过修改 vmx 文件实现自动映射，详细方法请看“将盒子自动映射入虚拟机中”）

如果需要 VMware 虚拟机启动时，自动将指定盒子映射入指定虚拟机里，请看“将盒子自动映射入虚拟机中”。

13.2. VirtualBox 虚拟机

13.2.1. 先设置虚拟机使能 USB

下图中选择 USB1.1 或 USB2.0 都是可以的

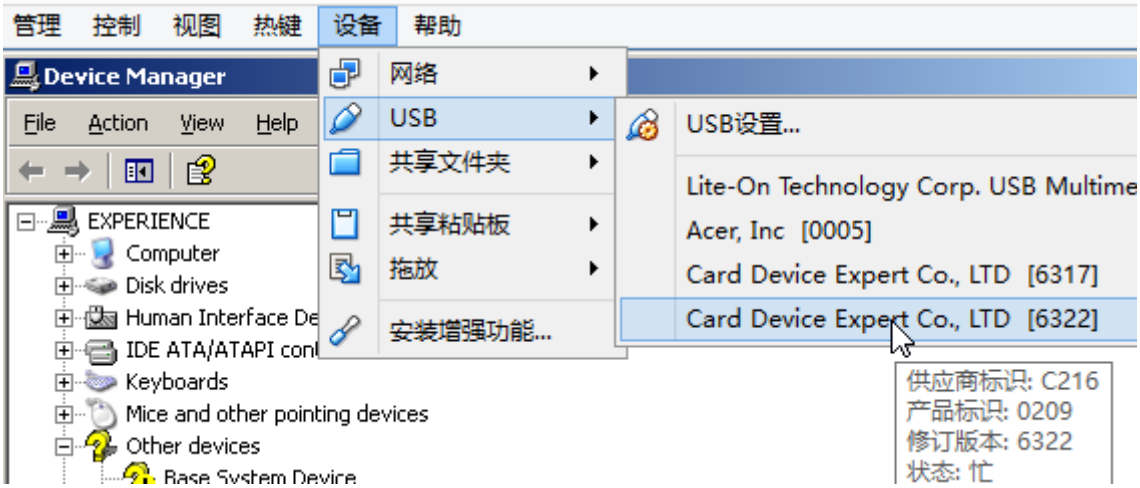


13.2.2. 启动虚拟机，进行映射

插上盒子，如果是双头盒子，请用 USB 延长线将两个 USB 头都连接到电脑。启动虚拟机，然后：

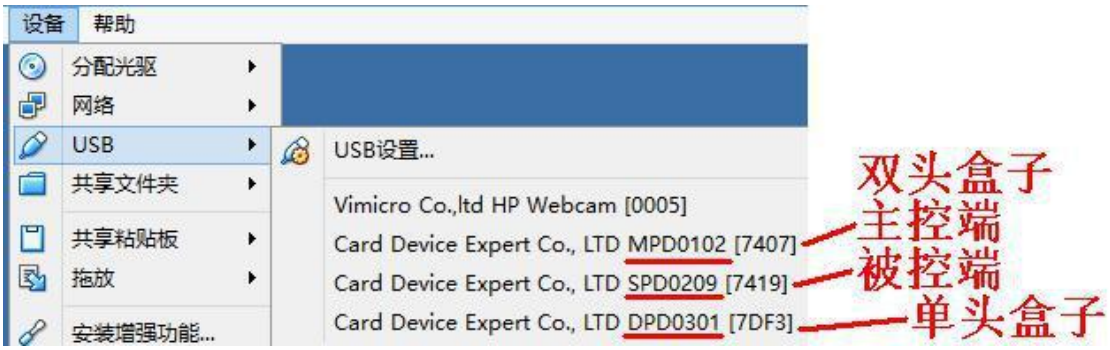
13.2.2.1. 无设备名时的单头盒子、双头盒子主控端和被控机

如下图设置映射：设备 -> USB 找到对应的设备（当鼠标放上去时，注意看鼠标下方显示的供应商标识和产品标识，单双头盒子供应商标识（VID）默认是 C216 或 C217；双头盒子被控端产品标识（PID）默认是 0209；双头盒子主控端 PID 默认是 0102；单头盒子 PID 默认是 0301），点击就可以映射。



13.2.2.2. 有设备名时的单头盒子、双头盒子主控端和被控机

如下图设置映射：设备 -> USB -> 含有设备名的字符串，点击就可以映射。



13.2.3. 如果设置正确，仍无法映射

请确认以下几点：

1、**你的虚拟机是否是买来的？**如果是买来的，而你又必须要用这个虚拟机，请联系卖主，让他把 USB 组件加进去。**对于这种虚拟机，请自行找虚拟机的卖家解决映射问题，我们不提供支持！**

如果不是必须要用买来的虚拟机，建议下载安装包，重新从头安装一个。

2、**盒子所插的 USB 口是否是 USB3.0？**USB3.0 口里面一般是蓝色的，USB2.0 口里面一般是黑色或者白色。如下图：

一些虚拟机不能支持 USB3.0 口，请将盒子插到 USB2.0 口上再尝试映射。

13.2.4. 如果插入多个盒子，只能成功映射一个或两个

VirtualBox 对多个具有相同硬件信息的 USB 设备映射存在问题，不能分别映射多个盒子到多个虚拟机中。

这个问题只能通过使用可修改 VID、PID 的盒子，将每个盒子的 VID、PID 设置成唯一的，这样就可以将多个盒子分别映射到多个虚拟机中。

而且，通过更改 VID、PID，还能使用虚拟机的筛选项，在启动虚拟机时，自动映射对应的盒子，省去手工映射的麻烦。请看下一节。

13.3. 将盒子自动映射入虚拟机中

用多个双头盒子实现一台主机控制多台虚拟机时，每次重启机器，都需要手工映射虚拟机，过程繁琐。

如果使用可修改 VID、PID 的双头盒子，则可以做到虚拟机启动后自动映射

指定盒子，省去映射的繁琐过程，同时主机中的脚本通过 M_Open_VidPid 接口也很容易就能区分到底操作的是哪个盒子、哪个虚拟机。

下述方法适用于可修改 VID、PID 的双头盒子，也适用于可修改 VID、PID 的单头盒子。以下仅以可修改 VID、PID 的双头盒子+主机控制虚拟机方式进行说明。

13.3.1. 先将每个盒子的 VID、PID 修改成唯一的

为了区分，建议：

1、将所有盒子的主控端 VID 都修改成一个值（如 C317），被控端 VID 也都修改成另一个值（如 C318）。

2、而每个盒子的主控端和被控端 PID 则都设置成相同的值。如：其中一个盒子的主控端和被控端 PID 都是 0001（在这个盒子表面可以贴一个标签 0001）；另一个盒子的的主控端和被控端 PID 都是 0002（在这个盒子表面可以贴一个标签 0002），以此类推。

如上修改后，主机上的脚本使用 M_Open_VidPid 打开端口时，如果代入的 PID 参数是 0001，则操作的盒子就是标签为 0001 的盒子，操作的虚拟机就是 0001 映射入的虚拟机。下面再说明如何将指定盒子映射到指定虚拟机中。

以上方案供参考，具体方案可以根据需要来定，只要保证每个盒子的 VID+PID 是唯一的。

13.3.2. VMware 虚拟机设置方法

1. 双头盒子被控端

在 VMware 配置文件（vmx 文件）中增加如下两行语句：

```
usb.quirks.device0 = "0xC318:0x0001 allow"
usb.autoConnect.device0 = "0xC318:0x0001"
```

就可以将 VID=C318、PID=0001 的双头盒子被控端映射入该虚拟机里。

2. 双头盒子主控端或单头盒子

对于双头盒子的主控端或者单头盒子，只需要在 VMware 配置文件（vmx 文件）中增加如下一行语句：

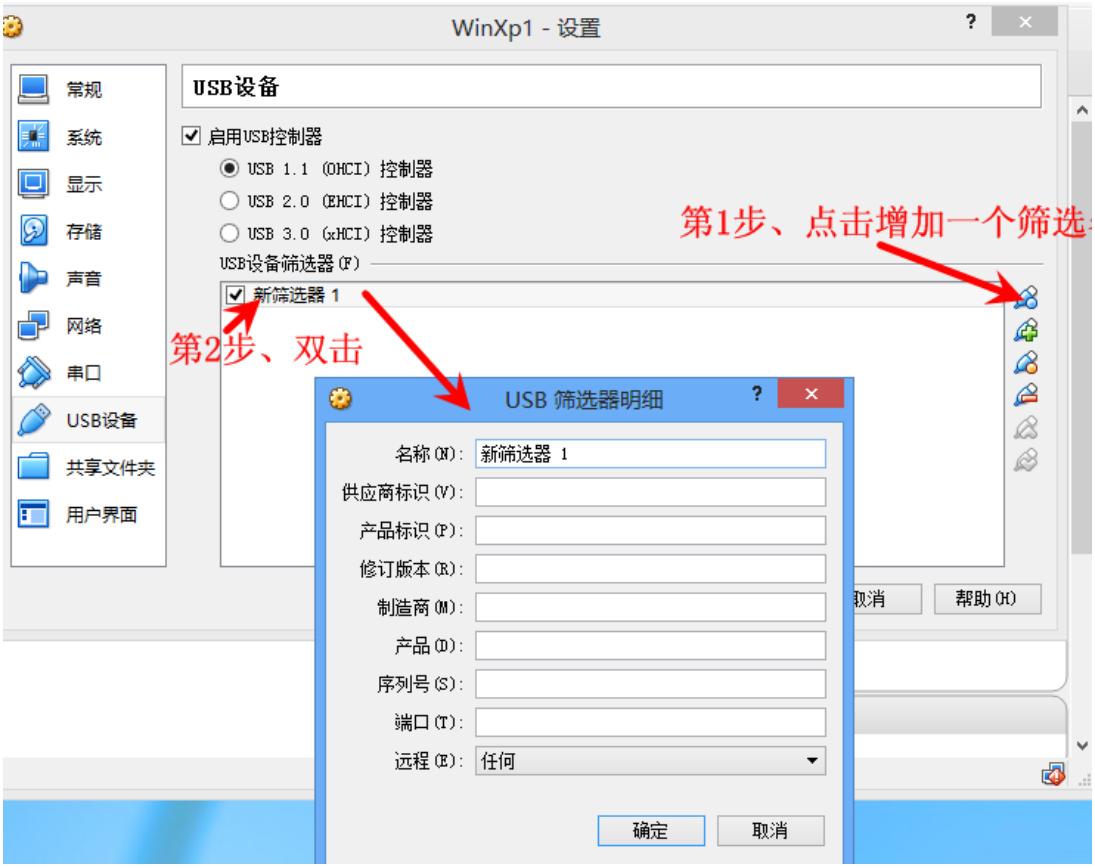
```
usb.autoConnect.device0 = "0xC317:0x0001"
```

就可以将 VID=C317、PID=0001 的双头盒子主控端映射入该虚拟机里。

注意：以上语句一定要在虚拟机刚创建时写入 vmx 文件。不要映射盒子后再写，否则有可能导致 vmx 文件损坏。如果出现 vmx 文件损坏无法打开虚拟机，请重新创建虚拟机，修改 vmx 文件后，再插入盒子进行映射。

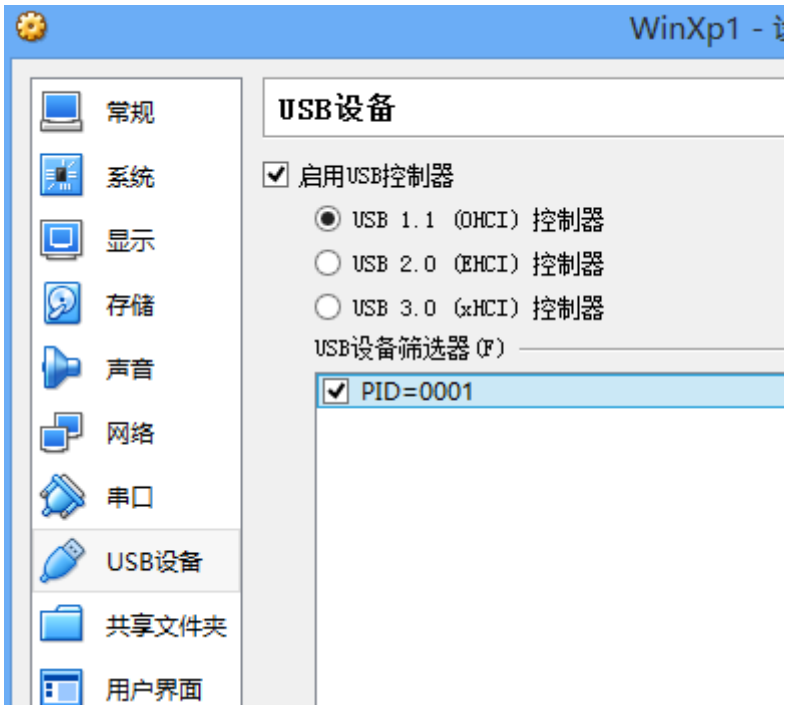
13.3.3. VirtualBox 虚拟机设置方法

1. 增加筛选器



2. 设置筛选器，点击确定

3. 设置完成



13.3.4. 在启动虚拟机前，请先将需要的盒子都插到电脑上，再启动虚拟机

如果一台电脑要实现一个主机控制 8 个虚拟机，则将这 8 个已经设置好 VID、PID 的盒子都插到电脑上（USB 口不够用，请用 USB Hub 集线器扩展 USB 口），然后再将已经设置好筛选器的虚拟机逐个启动。虚拟机启动后会自动将指定的盒子被控端映射到虚拟机中。

13.4. 关于虚拟机能支持的盒子数量

电脑本机理论上可以支持最大 126 个单头和或 63 个双头盒子（两个 USB 头都插到一台电脑上）。但是虚拟机并不能支持这么多盒子。

我们测过测试发现：在 64 位 Win7 + VMware 12 上，Vmware 最多只能开 26 个虚拟机，映射成功 26 个单头盒子。再多开虚拟机、多插盒子也无法映射成功。而 VirtualBox 最多只能开 20 个虚拟机，映射成功 20 个单头盒子。

以上测试结果仅供参考，并不代表所有机器所有系统都是这个结果。

13.5. 盒子映射入虚拟机后，会自动退出虚拟机的解决方法

在某些系统、机器上，盒子在映射入虚拟机后，隔段时间有个别盒子会自动

退出虚拟机。具体原因不明，可以按以下方法做一些尝试，不保证肯定能解决退出的问题。

首先请尽量做到以下几点：

1. 虚拟机是安装的，而不是购买的。
2. 盒子如果是插在 USB HUB 上，请先将 USB HUB 去掉，直接将盒子插到电脑的 USB 口上。
3. 虚拟机版本尽可能升级到最高版本。

以下的步骤都是在主机上操作。

13.5.1. 步骤 1，禁用 USB3.0

很多虚拟机不能支持 USB3.0，可以先将 USB3.0 禁用。有以下两种方法供选择：

13.5.1.1. 方法 1，在 BIOS 中禁用

1. 开机进入 BIOS
2. 在 BIOS 中 Config-USB-USB3.0 Mode 将默认的 Auto 选项更改为 Disabled 后再按 F10 保存即可

某些电脑的 BIOS 可能不具备上述功能，请尝试下面的方法 2。

13.5.1.2. 方法 2，在设备管理器中禁用

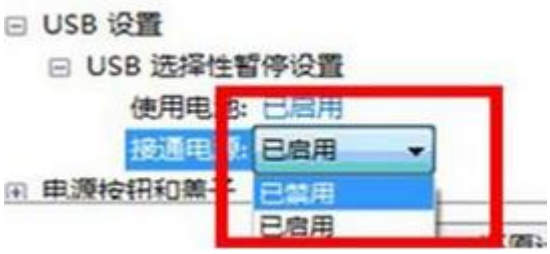
1. 进入“设备管理器”
2. 在“通用串行总线控制器”中找到 USB3.0 控制器，类似下图：

3. 在 usb 3.0 host controller 上点击右键选择禁用

13.5.2. 步骤 2，禁用 USB 休眠

首先禁止电脑休眠，然后再禁用 USB 休眠（WinXp 下无此功能，可以略过）。禁用方法如下：

1. 点击电源管理图标，选择“更多电源选项”，点击更改计划设置选项。
2. 点击“更改高级电源设置”选项
3. 选择“USB 选择性暂停设置”，然后设置为已禁用。如下图：



14. 双头盒子支持 Linux 系统

双头盒子的主控端、被控端都可以支持 Linux。绝对移动功能（M_MoveTo3 接口）也可以在 Linux 系统上使用。我们提供 Linux 接口源码，请自行移植编译。我们不提供技术支持。

采用 Linux 系统的彩票机、医疗设备、终端、各种 Pi 都可以支持。但是因为 Linux 设备种类众多，一切请以实际测试结果为准。

15. 双头盒子支持安卓系统

双头盒子可以实现电脑控制安卓平板/手机：只要将双头盒子的主控端接到电脑上，将被控端接到支持 OTG 功能的安卓平板/手机上，在电脑上运行脚本就可以。

由于安卓系统的鼠标移动比较特殊，要达到精确移动到指定坐标，脚本要满足以下几个条件：

1. 在刚打开句柄或者 ResetMousePos 后，脚本应用程序都先将鼠标移动到(0,0)，然后再移动到其他位置。
2. 每次移动，X 方向或者 Y 方向都不能大于 100。如果需要移动远距离，可以调用多次来移动，每次移动间隔至少 100ms。
3. 安卓系统设置中，“语言和输入法”->“指针速度”，要调到中间。

注意：绝对移动功能（M_MoveTo3 接口）不能在安卓系统上使用。

16. 技术支持

如果您有开发、使用方面的问题，可随时通过如下任一方式联系我们：
淘宝网站：http://freeeasy-life.taobao.com
QQ：2644439521

17. 附录 1：键盘码

名称	键盘码值（十进制）	对应实际键盘的按键
Keyboard_a	4	Keyboard a and A
Keyboard_b	5	Keyboard b and B
Keyboard_c	6	Keyboard c and C
Keyboard_d	7	Keyboard d and D
Keyboard_e	8	Keyboard e and E
Keyboard_f	9	Keyboard f and F
Keyboard_g	10	Keyboard g and G
Keyboard_h	11	Keyboard h and H
Keyboard_i	12	Keyboard i and I
Keyboard_j	13	Keyboard j and J
Keyboard_k	14	Keyboard k and K
Keyboard_l	15	Keyboard l and L
Keyboard_m	16	Keyboard m and M
Keyboard_n	17	Keyboard n and N
Keyboard_o	18	Keyboard o and O
Keyboard_p	19	Keyboard p and P
Keyboard_q	20	Keyboard q and Q
Keyboard_r	21	Keyboard r and R
Keyboard_s	22	Keyboard s and S
Keyboard_t	23	Keyboard t and T
Keyboard_u	24	Keyboard u and U
Keyboard_v	25	Keyboard v and V
Keyboard_w	26	Keyboard w and W
Keyboard_x	27	Keyboard x and X
Keyboard_y	28	Keyboard y and Y
Keyboard_z	29	Keyboard z and Z
Keyboard_1	30	Keyboard 1 and !
Keyboard_2	31	Keyboard 2 and @
Keyboard_3	32	Keyboard 3 and #

Keyboard_4	33	Keyboard 4 and $
Keyboard_5	34	Keyboard 5 and %
Keyboard_6	35	Keyboard 6 and ^
Keyboard_7	36	Keyboard 7 and &
Keyboard_8	37	Keyboard 8 and *
Keyboard_9	38	Keyboard 9 and (
Keyboard_0	39	Keyboard 0 and)
Keyboard_ENTER	40	Keyboard ENTER
Keyboard_ESCAPE	41	Keyboard ESCAPE
Keyboard_Backspace	42	Keyboard Backspace
Keyboard_Tab	43	Keyboard Tab
Keyboard_KongGe	44	Keyboard Spacebar
Keyboard_JianHao	45	Keyboard - and _(underscore)
Keyboard_DengHao	46	Keyboard = and +
Keyboard_ZuoZhongKuoHao	47	Keyboard [and {
Keyboard_YouZhongKuoHao	48	Keyboard] and }
Keyboard_FanXieGang	49	Keyboard \ and |
Keyboard_FenHao	51	Keyboard ; and :
Keyboard_DanYinHao	52	Keyboard ‘ and “
Keyboard_BoLangXian	53	Keyboard `(Grave Accent) and ~(Tilde)
Keyboard_Douhao	54	Keyboard, and <
Keyboard_JuHao	55	Keyboard . and >
Keyboard_XieGang_WenHao	56	Keyboard / and ?
Keyboard_CapsLock	57	Keyboard Caps Lock
Keyboard_F1	58	Keyboard F1
Keyboard_F2	59	Keyboard F2
Keyboard_F3	60	Keyboard F3
Keyboard_F4	61	Keyboard F4
Keyboard_F5	62	Keyboard F5
Keyboard_F6	63	Keyboard F6
Keyboard_F7	64	Keyboard F7
Keyboard_F8	65	Keyboard F8
Keyboard_F9	66	Keyboard F9
Keyboard_F10	67	Keyboard F10
Keyboard_F11	68	Keyboard F11
Keyboard_F12	69	Keyboard F12
Keyboard_PrintScreen	70	Keyboard PrintScreen
Keyboard_ScrollLock	71	Keyboard Scroll Lock
Keyboard_Pause	72	Keyboard Pause
Keyboard_Insert	73	Keyboard Insert
Keyboard_Home	74	Keyboard Home
Keyboard_PageUp	75	Keyboard PageUp
Keyboard_Delete	76	Keyboard Delete

Keyboard_End	77	Keyboard End
Keyboard_PageDown	78	Keyboard PageDown
Keyboard_RightArrow	79	Keyboard RightArrow
Keyboard_LeftArrow	80	Keyboard LeftArrow
Keyboard_DownArrow	81	Keyboard DownArrow
Keyboard_UpArrow	82	Keyboard UpArrow
Keypad_NumLock	83	Keypad Num Lock and Clear
Keypad_ChuHao	84	Keypad /
Keypad_ChengHao	85	Keypad *
Keypad_JianHao	86	Keypad -
Keypad_JiaHao	87	Keypad +
Keypad_ENTER	88	Keypad ENTER
Keypad_1_and_End	89	Keypad 1 and End
Keypad_2_and_DownArrow	90	Keypad 2 and Down Arrow
Keypad_3_and_PageDn	91	Keypad 3 and PageDn
Keypad_4_and_LeftArrow	92	Keypad 4 and Left Arrow
Keypad_5	93	Keypad 5
Keypad_6_and_RightArrow	94	Keypad 6 and Right Arrow
Keypad_7_and_Home	95	Keypad 7 and Home
Keypad_8_and_UpArrow	96	Keypad 8 and Up Arrow
Keypad_9_and_PageUp	97	Keypad 9 and PageUp
Keypad_0_and_Insert	98	Keypad 0 and Insert
Keypad_Dian_and_Delete	99	Keypad . and Delete
Keyboard_Application	101	Keyboard Application
Keyboard_LeftControl	224	Keyboard Left Ctrl
Keyboard_LeftShift	225	Keyboard Left Shift
Keyboard_LeftAlt	226	Keyboard Left Alt
Keyboard_LeftWindows	227	Keyboard Left Windows
Keyboard_RightControl	228	Keyboard Right Ctrl
Keyboard_RightShift	229	Keyboard Right Shift
Keyboard_RightAlt	230	Keyboard Right Alt
Keyboard_RightWindows	231	Keyboard Right Windows

18. 附录 2：VK 键值

名称	VK 键盘码值（十进制）	对应实际键盘上的按键
VK_BACK	8	Backspace
VK_TAB	9	Tab
VK_RETURN	13	Enter
VK_SHIFT	16	Shift

VK_CONTROL	17	Ctrl
VK_MENU	18	Alt
VK_PAUSE	19	Pause
VK_CAPITAL	20	Caps Lock
VK_ESCAPE	27	Esc
VK_SPACE	32	Space
VK_PRIOR	33	Page Up
VK_NEXT	34	Page Down
VK_END	35	End
VK_HOME	36	Home
VK_LEFT	37	Left Arrow
VK_UP	38	Up Arrow
VK_RIGHT	39	Right Arrow
VK_DOWN	40	Down Arrow
VK_SNAPSHOT	44	Print Screen
VK_INSERT	45	Insert
VK_DELETE	46	Delete
	48	0
	49	1
	50	2
	51	3
	52	4
	53	5
	54	6
	55	7
	56	8
	57	9
	65	A
	66	B
	67	C
	68	D
	69	E
	70	F
	71	G
	72	H
	73	I
	74	J
	75	K
	76	L

	77	M
	78	N
	79	O
	80	P
	81	Q
	82	R
	83	S
	84	T
	85	U
	86	V
	87	W
	88	X
	89	Y
	90	Z
VK_LWIN	91	Left Windows
VK_RWIN	92	Right Windows
VK_NUMPAD0	96	小键盘 0
VK_NUMPAD1	97	小键盘 1
VK_NUMPAD2	98	小键盘 2
VK_NUMPAD3	99	小键盘 3
VK_NUMPAD4	100	小键盘 4
VK_NUMPAD5	101	小键盘 5
VK_NUMPAD6	102	小键盘 6
VK_NUMPAD7	103	小键盘 7
VK_NUMPAD8	104	小键盘 8
VK_NUMPAD9	105	小键盘 9
VK_MULTIPLY	106	小键盘 *
VK_ADD	107	小键盘 +
VK_SEPARATOR	108	小键盘 Enter
VK_SUBTRACT	109	小键盘 -
VK_DECIMAL	110	小键盘 .
VK_DIVIDE	111	小键盘 /
VK_F1	112	F1
VK_F2	113	F2
VK_F3	114	F3
VK_F4	115	F4
VK_F5	116	F5
VK_F6	117	F6
VK_F7	118	F7

VK_F8	119	F8
VK_F9	120	F9
VK_F10	121	F10
VK_F11	122	F11
VK_F12	123	F12
VK_NUMLOCK	144	Num Lock
VK_SCROLL	145	Scroll
VK_LSHIFT	160	Left Shift
VK_RSHIFT	161	Right Shift
VK_LCONTROL	162	Left Control
VK_RCONTROL	163	Right Control
VK_LMENU	164	Left Alt
VK_RMENU	165	Right Alt
VK_OEM_1	186	; :
VK_OEM_PLUS	187	= +
VK_OEM_COMMA	188	,<
VK_OEM_MINUS	189	- _
VK_OEM_PERIOD	190	.>
VK_OEM_2	191	/ ?
VK_OEM_3	192	` ~
VK_OEM_4	219	[{
VK_OEM_5	220	\ |
VK_OEM_6	221	] }
VK_OEM_7	222	’ ″