

**D2XX 程序员指南**

### 1.5版

#### 发布日期：2023年7月9日

FTDI 为其驱动程序提供 DLL 和虚拟 COM 端口 (VCP) 应用程序接口。本文档提供 FTD2XX DLL 函数库的应用程序编程接口 (API)。

在生命支持和/或安全应用中使用 FTDI 设备的风险完全由用户自行承担，并且用户同意为 FTDI 辩护、赔偿并使 FTDI 免受因此类使用而造成的任何及所有损害、索赔、诉讼或费用。

未来技术设备国际有限公司 (FTDI) Unit 1, 2 Seaward Place, Glasgow G41 1HH, United Kingdom 电话：+44 (0) 141 第429章 2777 传真：+ 44 (0) 141 429 2758

网站： [http://ftdichip.com](http://ftdichip.com/)

版权所有 © 未来科技设备国际有限公司

**目录**

1. [前言](#_bookmark0) [5](#_bookmark0)
2. [简介](#_bookmark1) [6](#_bookmark1)
3. [D2XX 经典的 功能](#_bookmark3) [7](#_bookmark3)
   1. [FT\_设置VIDPID](#_bookmark4)  [7](#_bookmark4)
   2. [FT\_GetVIDPID](#_bookmark5)  [7](#_bookmark5)
   3. [FT\_创建设备信息列表](#_bookmark6) [8](#_bookmark6)
   4. [FT\_获取设备信息列表](#_bookmark7) [9](#_bookmark7)
   5. [FT\_GetDeviceInfoDetail](#_bookmark8)  [10](#_bookmark8)
   6. [FT\_列表设备](#_bookmark9) [12](#_bookmark9)
   7. [FT\_打开](#_bookmark10) [14](#_bookmark10)
   8. [FT\_OpenEx](#_bookmark11)  [15](#_bookmark11)
   9. [FT\_关闭](#_bookmark12) [18](#_bookmark12)
   10. [FT\_读](#_bookmark13) [18](#_bookmark13)
   11. [FT\_写](#_bookmark14) [20](#_bookmark14)
   12. [FT\_设置波特率](#_bookmark15) [21](#_bookmark15)
   13. [FT\_设置除数](#_bookmark16) [22](#_bookmark16)
   14. [FT\_SetDataCharacteristics](#_bookmark17)  [22](#_bookmark17)
   15. [FT\_设置超时](#_bookmark18) [23](#_bookmark18)
   16. [FT\_设置流量控制](#_bookmark19) [24](#_bookmark19)
   17. [FT\_设置Dtr](#_bookmark20)  [25](#_bookmark20)
   18. [FT\_ClrDtr](#_bookmark21)  [26](#_bookmark21)
   19. [FT\_设置Rts](#_bookmark22)  [27](#_bookmark22)
   20. [FT\_ClrRts](#_bookmark23)  [28](#_bookmark23)
   21. [FT\_GetModemStatus](#_bookmark24)  [28](#_bookmark24)
   22. [FT\_GetQueueStatus](#_bookmark26)  [29](#_bookmark26)
   23. [FT\_获取设备信息](#_bookmark27) [30](#_bookmark27)
   24. [FT\_GetDriver版本](#_bookmark28) [32](#_bookmark28)
   25. [FT\_GetLibrary版本](#_bookmark29) [33](#_bookmark29)
   26. [FT\_GetComPortNumber](#_bookmark30)  [33](#_bookmark30)
   27. [FT\_获取状态](#_bookmark31) [34](#_bookmark31)
   28. [FT\_SetEventNotification](#_bookmark32)  [35](#_bookmark32)
   29. [FT\_设置字符](#_bookmark33) [37](#_bookmark33)
   30. [FT\_SetBreakOn](#_bookmark34)  [38](#_bookmark34)
   31. [FT\_SetBreakOff](#_bookmark35)  [39](#_bookmark35)
   32. [FT\_清除](#_bookmark36) [39](#_bookmark36)
   33. [FT\_重置设备](#_bookmark37) [40](#_bookmark37)
   34. [FT\_重置端口](#_bookmark38) [41](#_bookmark38)
   35. [FT\_循环端口](#_bookmark39) [41](#_bookmark39)
   36. [FT\_重新扫描](#_bookmark40) [42](#_bookmark40)
   37. [FT\_重新加载](#_bookmark41) [43](#_bookmark41)
   38. [FT\_SetResetPipeRetryCount](#_bookmark42)  [44](#_bookmark42)
   39. [FT\_StopInTask](#_bookmark43)  [44](#_bookmark43)
   40. [FT\_RestartInTask](#_bookmark44)  [45](#_bookmark44)
   41. [FT\_SetDeadman超时](#_bookmark45) [46](#_bookmark45)
   42. [FT\_IoCtl](#_bookmark46)  [47](#_bookmark46)
   43. [FT\_SetWaitMask](#_bookmark47)  [47](#_bookmark47)
   44. [FT\_等待掩码](#_bookmark48) [47](#_bookmark48)
4. [EEPROM编程 界面 功能](#_bookmark49) [48](#_bookmark49)
   1. [FT\_ReadEE](#_bookmark50)  [48](#_bookmark50)
   2. [FT\_WriteEE](#_bookmark51)  [49](#_bookmark51)
   3. [FT\_EraseEE](#_bookmark52)  [49](#_bookmark52)
   4. [FT\_EE\_读取](#_bookmark53) [50](#_bookmark53)
   5. [FT\_EE\_ReadEx](#_bookmark54)  [51](#_bookmark54)
   6. [FT\_EE\_程序](#_bookmark55) [52](#_bookmark55)
   7. [FT\_EE\_ProgramEx](#_bookmark56)  [55](#_bookmark56)
   8. [FT\_EE\_UAS尺寸](#_bookmark57) [56](#_bookmark57)
   9. [FT\_EE\_UA读](#_bookmark58) [57](#_bookmark58)
   10. [FT\_EE\_UAWrite](#_bookmark59)  [58](#_bookmark59)
   11. [FT\_EEPROM\_读取](#_bookmark60) [59](#_bookmark60)
   12. [FT\_EEPROM\_程序](#_bookmark61) [61](#_bookmark61)
5. [扩展 应用程序编程接口 功能](#_bookmark62) [64](#_bookmark62)
   1. [FT\_设置延迟计时器](#_bookmark63) [64](#_bookmark63)
   2. [FT\_GetLatencyTimer](#_bookmark64)  [65](#_bookmark64)
   3. [FT\_设置位模式](#_bookmark65) [66](#_bookmark65)
   4. [FT\_GetBitMode](#_bookmark66)  [67](#_bookmark66)
   5. [FT\_设置USB参数](#_bookmark67) [68](#_bookmark67)
6. [FT-Win32 应用程序编程接口 功能](#_bookmark68) [70](#_bookmark68)
   1. [FT\_W32\_创建文件](#_bookmark69) [70](#_bookmark69)
   2. [FT\_W32\_CloseHandle](#_bookmark70)  [72](#_bookmark70)
   3. [FT\_W32\_读取文件](#_bookmark71) [72](#_bookmark71)
   4. [FT\_W32\_WriteFile](#_bookmark72)  [75](#_bookmark72)
   5. [FT\_W32\_GetOverlappedResult](#_bookmark73)  [77](#_bookmark73)
   6. [FT\_W32\_EscapeCommFunction](#_bookmark74)  [77](#_bookmark74)
   7. [FT\_W32\_GetCommModemStatus](#_bookmark75)  [78](#_bookmark75)
   8. [FT\_W32\_SetupComm](#_bookmark76)  [79](#_bookmark76)
   9. [FT\_W32\_SetCommState](#_bookmark77)  [80](#_bookmark77)
   10. [FT\_W32\_GetCommState](#_bookmark78)  [80](#_bookmark78)
   11. [FT\_W32\_SetCommTimeouts](#_bookmark79)  [81](#_bookmark79)
   12. [FT\_W32\_GetCommTimeouts](#_bookmark80)  [82](#_bookmark80)
   13. [FT\_W32\_SetCommBreak](#_bookmark81)  [83](#_bookmark81)
   14. [FT\_W32\_ClearCommBreak](#_bookmark82)  [84](#_bookmark82)
   15. [FT\_W32\_SetCommMask](#_bookmark83)  [84](#_bookmark83)
   16. [FT\_W32\_GetCommMask](#_bookmark84)  [85](#_bookmark84)
   17. [FT\_W32\_WaitCommEvent](#_bookmark85)  [86](#_bookmark85)
   18. [FT\_W32\_PurgeComm](#_bookmark86)  [88](#_bookmark86)
   19. [FT\_W32\_GetLastError](#_bookmark87)  [89](#_bookmark87)
   20. [FT\_W32\_ClearCommError](#_bookmark88)  [89](#_bookmark88)
7. [接触 信息](#_bookmark89) [91](#_bookmark89)

[附录 A - 类型 定义](#_bookmark90) [92](#_bookmark90)

[附录B – 参考文献](#_bookmark104) [106](#_bookmark104)

[文档 参考文献](#_bookmark105) [106](#_bookmark105)

[缩略语 和 缩写](#_bookmark106) [106](#_bookmark106)

[附录 C – 表格列表 & 图](#_bookmark107) [107](#_bookmark107)

[列表 的 表](#_bookmark108) [107](#_bookmark108)

[列表 的 图](#_bookmark109) [107](#_bookmark109)

[附录 D - 修订 历史](#_bookmark110) [108](#_bookmark110)

# 前言

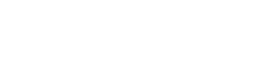
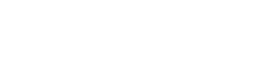
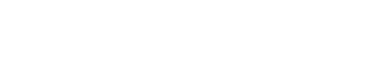
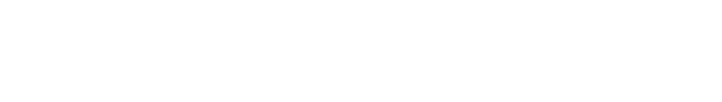
D2XX 接口是专门用于 FTDI 设备的专有接口。本文档解释了应用程序开发人员可通过 FTD2XX 库使用的功能。

本文档中给出的任何软件代码示例仅供参考。这些示例不受 FTDI 的保证和支持。

# 介绍

FTDI 为其 USB-UART 和 USB-FIFO IC 系列提供两种替代软件接口。其中一个接口提供虚拟 COM 端口 (VCP)，该端口在系统中显示为传统 COM 端口。第二个接口 D2XX 通过专有 DLL (FTD2XX.DLL) 提供。 D2XX 接口提供标准操作系统 COM 端口 API 中不可用的特殊功能，例如将设备设置为不同模式或将数据写入设备 EEPROM。

对于 Windows 的 FTDI 驱动程序，D2XX 驱动程序和 VCP 驱动程序分布在同一个驱动程序包中，称为组合驱动程序模型 (CDM) 包。[图 2.1 Windows CDM 驱动程序](#_bookmark2) [体系结构](#_bookmark2)说明了 Windows CDM 驱动程序的体系结构。



**D2XX Application**

**COM Port Application**

**FTD2XX.DLL**

**FTSER2K.SYS**

**FTDIBUS.SYS**

**USB Host Controller Driver**

#### 图 2.1 Windows CDM 驱动程序架构

对于 Linux、Mac OS X（10.4 及更高版本）和 Windows CE（4.2 及更高版本），D2XX 驱动程序和 VCP 驱动程序是互斥的选项，因为在给定时间对于给定设备 ID 只能安装一种驱动程序类型。对于运行 CDM 驱动程序的 Windows 系统，应用程序可以使用 D2XX 或 VCP 接口，而无需安装不同的驱动程序，但不能同时使用这两个接口。

由于 VCP 驱动程序接口旨在模拟传统 COM 端口，因此 FTDI 不提供有关如何从应用程序与 VCP 驱动程序进行通信的文档；开发人员参考了互联网上有关串行通信的大量可用材料。

D2XX 接口是专门用于 FTDI 设备的专有接口。本文档解释了应用程序开发人员可通过 FTD2XX 库使用的功能。

# D2XX 经典 功能

本节列出的功能与所有 FTDI 设备兼容。

## FT\_设置VIDPID

支持的操作系统 Linux

Mac OS X（10.4 及更高版本）

### 概括

用于在内部设备列表中包含自定义 VID 和 PID 组合的命令。这将允许驱动程序加载指定的 VID 和 PID 组合。

### 定义

FT\_STATUS **FT\_SetVIDPID** (DWORD *dwVID* , DWORD *dwPID* )

### 参数

*dwVID* 设备供应商 ID （视频）

*dwPID* 设备产品 ID (PID)

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

默认情况下，驱动程序将支持一组有限的 VID 和 PID 匹配设备（仅限 VID 0x0403 和 PID 0x6001、0x6010、0x6006）。

为了将驱动程序与其他 VID 和 PID 组合一起使用，必须在调用[FT\_ListDevices](#_bookmark9) 、 [FT\_Open](#_bookmark10) 、 [FT\_OpenEx](#_bookmark11)或[FT\_CreateDeviceInfoList之前使用 FT\_SetVIDPID 函数](#_bookmark6)。

## FT\_获取VIDPID

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本）

### 概括

从内部设备列表中检索当前 VID 和 PID 组合的命令。

### 定义

FT\_STATUS **FT\_GetVIDPID** (DWORD \* *pdwVID* , DWORD \* *pdwPID* )

### 参数

*pdwVID* 指向将包含内部数据的 DWORD 的指针 视频识别码

*pdwPID* 指向将包含内部的 DWORD 的指针 PID

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

请参阅[FT\_SetVIDPID](#_bookmark4) 。

## FT\_创建设备信息列表

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

该函数构建设备信息列表并返回连接到系统的 D2XX 设备的数量。该列表包含有关未打开和打开的设备的信息。

### 定义

FT\_STATUS **FT\_CreateDeviceInfoList** (LPDWORD *lpdwNumDevs* )

### 参数

*lpdwNumDevs* 指向 unsigned long 的指针，用于存储数字 的 连接的设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

应用程序可以使用此函数来获取连接到系统的设备数量。然后，它可以为设备信息列表分配空间[，并使用](#_bookmark8)[FT\_GetDeviceInfoList](#_bookmark7)或[FT\_GetDeviceInfoDetail FT\_GetDeviceInfoDetail](#_bookmark8)检索列表。

如果连接到系统的设备发生变化，设备信息列表将不会更新，直到再次调用 FT\_CreateDeviceInfoList。

### 例子

FT\_STATUS ft状态；

DWORD numDevs；

// 创建设备信息列表 ftStatus = FT\_CreateDeviceInfoList(&numDevs); if (ftStatus == FT\_OK) {

printf("设备数量为 %d\n",numDevs);

}

别的 {

}

// FT\_CreateDeviceInfoList 失败

## FT\_获取设备信息列表

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数返回设备信息列表以及列表中D2XX设备的数量。

### 定义

FT\_STATUS **FT\_GetDeviceInfoList** (FT\_DEVICE\_LIST\_INFO\_NODE \* *pDest* ,

LPDWORD *lpdwNumDevs* )

### 参数

*\*pDest指向* [FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102)数组的指针结构。

*lpdwNumDevs* 指向元素数量的指针 大批。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

[FT\_CreateDeviceInfoList](#_bookmark6)之后调用此函数。如果连接到系统的设备发生变化，设备信息列表将不会更新，直到再次调用[FT\_CreateDeviceInfoList 。](#_bookmark6)

调用[FT\_CreateDeviceInfoList](#_bookmark6)时，不会返回打开的设备的位置 ID 信息。对于在其他进程中打开的设备，信息不可用。在这种情况下，*标志*

[FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102)参数将指示设备已打开，但其他字段将指示

无人居住。

标志值是一个 4 字节位图，包含[附录 A – 类型中定义的杂项数据](#_bookmark103) [定义](#_bookmark103)。该数字的位 0（最低有效位）指示端口是打开 (1) 还是关闭 (0)。位 1 指示设备是否被枚举为高速 USB 设备 (2) 还是全速 USB 设备 (0)。其余位 (2 - 31) 被保留。

[FT\_DEVICE\_LIST\_INFO\_NODES](#_bookmark102)数组包含每个设备上的所有可用数据。 [FT\_DEVICE\_LIST\_INFO\_NODES](#_bookmark102)的结构在附录中给出。列表的存储空间必须由应用程序分配。 [FT\_CreateDeviceInfoList](#_bookmark6)返回的设备数量可用于执行此操作。

当使用 Visual Basic、LabVIEW 或类似语言进行编程时，可能需要[FT\_GetDeviceInfoDetail](#_bookmark8)而不是此函数。

请注意，Linux、Mac OS X 和 Windows CE 不支持位置 ID。因此，在这些操作系统下，结构中的位置 ID 参数将为空。

#### 例子

FT\_STATUS ft状态； FT\_DEVICE\_LIST\_INFO\_NODE \*devInfo; DWORD numDevs；

// 创建设备信息列表 ftStatus = FT\_CreateDeviceInfoList(&numDevs);

如果（ftStatus == FT\_OK）{

printf("设备数量为 %d\n",numDevs);

}

if (numDevs > 0) {

// 根据 numDevs devInfo = 为列表分配存储空间

(FT\_DEVICE\_LIST\_INFO\_NODE\*)malloc(sizeof(FT\_DEVICE\_LIST\_INFO\_NODE)\*numDevs);

//获取设备信息列表

ftStatus = FT\_GetDeviceInfoList(devInfo,&numDevs);如果（ftStatus == FT\_OK）{

for (int i = 0; i < numDevs; i++) { printf("Dev %d:\n",i); }

printf(" 标志=0x%x\n",devInfo[i].Flags); printf(" 类型=0x%x\n",devInfo[i].Type); printf(" ID=0x%x\n",devInfo[i].ID); printf(" LocId=0x%x\n",devInfo[i].LocId);

printf(" 序列号=%s\n",devInfo[i].序列号); printf(" 描述=%s\n",devInfo[i].Description); printf(" ftHandle=0x%x\n",devInfo[i].ftHandle);

}

}

}

## FT\_GetDeviceInfoDetail 获取设备信息详细信息

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数返回设备信息列表中的条目。

### 定义

FT\_STATUS **FT\_GetDeviceInfoDetail** (DWORD *dwIndex* , LPDWORD *lpdwFlags* ,

LPDWORD *lpdw类型*,

LPDWORD *lpdwID* 、LPDWORD *lpdwLocId* 、PCHAR pcSerialNumber、PCHAR pcDescription、FT\_HANDLE *\*ftHandle* ）

### 参数

*dwIndex* 设备信息中条目的索引 列表。

*lpdwFlags* 指向 unsigned long 的指针，用于存储标志 价值。

*lpdwType* 指向 unsigned long 的指针以存储设备 类型。

*lpdwID* 指向 unsigned long 的指针以存储设备 ID。

*lpdwLocId* 指向 unsigned long 的指针，用于存储设备位置 ID。

*pcSerialNumber* 指向缓冲区的指针，用于将设备序列号存储为 无效的-

终止的字符串。

*pcDescription* 指向缓冲区的指针，用于将设备描述存储为 空终止

细绳。

*\*ftHandle* 指向 FT\_HANDLE 类型变量的指针，其中句柄 将要 被存储。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

[FT\_CreateDeviceInfoList](#_bookmark6)之后调用此函数。如果连接到系统的设备发生变化，设备信息列表将不会更新，直到再次调用[FT\_CreateDeviceInfoList 。](#_bookmark6)

索引值从零开始。

标志值是一个 4 字节位图，包含[附录 A – 类型中定义的杂项数据](#_bookmark103) [定义](#_bookmark103)。该数字的位 0（最低有效位）指示端口是打开 (1) 还是关闭 (0)。位 1 指示设备是否被枚举为高速 USB 设备 (2) 还是全速 USB 设备 (0)。其余位 (2 - 31) 被保留。

调用[FT\_CreateDeviceInfoList](#_bookmark6)时，不会返回打开的设备的位置 ID 信息。对于在其他进程中打开的设备，信息不可用。在这种情况下， *lpdwFlags*

参数将指示设备已打开，但其他字段将不填充。

[FT\_DEVICE\_LIST\_INFO\_NODE结构](#_bookmark102)数组返回，请使用[FT\_CreateDeviceInfoList](#_bookmark6) 。

请注意，Linux、Mac OS X 和 Windows CE 不支持位置 ID。因此，在这些操作系统下，结构中的位置 ID 参数将为空。

### 例子

FT\_STATUS ft状态；

FT\_HANDLE ftHandleTemp；

DWORD numDevs；

双字标志；

双字 ID；

双字类型；

DWORD 本地 ID；

字符序列号[16]；字符描述[64]；

// 创建设备信息列表 ftStatus = FT\_CreateDeviceInfoList(&numDevs); if (ftStatus == FT\_OK) {

printf("设备数量为 %d\n",numDevs);

}

if (numDevs > 0) {

//获取设备0的信息

ftStatus = FT\_GetDeviceInfoDetail(0, &Flags, &Type, &ID, &LocId, 序列号, 描述, &ftHandleTemp);

if (ftStatus == FT\_OK) { printf("Dev 0:\n");

printf(" 标志=0x%x\n",标志); printf(" 类型=0x%x\n",类型); printf(" ID=0x%x\n",ID); printf(" LocId=0x%x\n",LocId);

printf(" 序列号=%s\n",序列号); printf(" 描述=%s\n",描述); printf(" ftHandle=0x%x\n",ftHandleTemp);

}

}

## FT\_列表设备

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取有关当前连接的设备的信息。该函数可以返回这样的信息

如连接的设备数量、设备序列号和设备描述字符串以及连接设备的位置 ID。

### 定义

FT\_STATUS **FT\_ListDevices** （PVOID *pvArg1* 、PVOID *pvArg2* 、DWORD *dwFlags* ）

### 参数

*pvArg1* 含义取决于 dwFlags。

*pvArg2* 含义取决于 dwFlags。

*dwFlags* 确定返回的格式 信息。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数可以通过多种方式使用来返回不同类型的信息。获取设备信息的更强大方法是使用[FT\_CreateDeviceInfoList](#_bookmark6) 、 [FT\_GetDeviceInfoList](#_bookmark7)和[FT\_GetDeviceInfoDetail](#_bookmark8)函数，因为它们返回设备上的所有可用信息。

最简单的形式是，它可用于返回当前连接的设备数量。如果*dwFlags*中设置了[*FT\_LIST\_NUMBER\_ONLY位*](#_bookmark92)，则参数*pvArg1*被解释为指向 DWORD 位置的指针，用于存储当前连接的设备数量。

它可用于返回设备信息：如果*dwFlags*中设置了[*FT\_OPEN\_BY\_SERIAL\_NUMBER位*](#_bookmark93)，则序列号字符串将

被退回；如果*dwFlags*中设置了[*FT\_OPEN\_BY\_DESCRIPTION位*](#_bookmark93)，则将返回产品描述字符串；如果*dwFlags*中设置了[*FT\_OPEN\_BY\_LOCATION位*](#_bookmark93)，则将返回位置 ID；如果这些位均未设置，则默认返回序列号字符串。

它可用于返回单个设备的设备字符串信息。如果dwFlags 中设置了[*FT\_LIST\_BY\_INDEX*](#_bookmark92)和[*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93)或[*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93)位，则参数 pvArg1 被解释为设备索引，参数 pvArg2 被解释为指向包含适当字符串的缓冲区的指针。索引从零开始，对于无效索引，将返回错误代码[*FT\_DEVICE\_NOT\_FOUND 。*](#_bookmark91)

它可用于返回所有连接设备的设备字符串信息。如果[*FT\_LIST\_ALL*](#_bookmark92)并且[*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93)或[*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93)位在 dwFlags 中设置，参数 pvArg1 被解释为指向缓冲区的指针数组的指针，以包含适当的字符串，参数 pvArg2 被解释为指向 DWORD 位置的指针，以存储当前设备的数量连接的。请注意，对于 pvArg1，指向缓冲区的指针数组中的最后一个条目应该是 NULL 指针，因此该数组将包含比连接的设备数量多一个的位置。

*dwFlags*中设置了[*FT\_LIST\_BY\_INDEX*](#_bookmark92)和[*FT\_OPEN\_BY\_LOCATION位*](#_bookmark93)，则返回设备的位置 ID 。在这种情况下，参数*pvArg1*被解释为设备的索引，参数*pvArg2*被解释为指向包含位置 ID 的 long 类型变量的指针。索引从零开始，对于无效索引，将返回错误代码[*FT\_DEVICE\_NOT\_FOUND 。*](#_bookmark91)请注意，Windows CE 和 Linux 不支持位置 ID。

*dwFlags*中设置了[*FT\_LIST\_ALL*](#_bookmark92)和[*FT\_OPEN\_BY\_LOCATION位*](#_bookmark93)，则返回所有连接设备的位置 ID 。在这种情况下，参数*pvArg1*被解释为指向 long 类型变量数组的指针，以包含位置 ID，参数*pvArg2*被解释为指向 DWORD 位置的指针，以存储当前连接的设备数量。

### 例子

下面的示例使用这些变量。

FT\_STATUS ft状态；

DWORD numDevs；

1. 获取当前设备数量 连接的

ftStatus = FT\_ListDevices(&numDevs,NULL,FT\_LIST\_NUMBER\_ONLY);如果（ftStatus == FT\_OK）{

// FT\_ListDevices OK，连接的设备数量以 numDevs 为单位

}

别的 {

}

// FT\_ListDevices 失败

1. 获取第一个序列号 设备

DWORD devIndex = 0; // 第一个设备

字符缓冲区[64]； // 空间足够了！

ftStatus = FT\_ListDevices((PVOID)devIndex,缓冲区,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_SERIAL\_NUMBER);如果（ftStatus == FT\_OK）{

// FT\_ListDevices OK，序列号在Buffer中

}

别的 {

}

// FT\_ListDevices 失败

请注意，索引是从零开始的。如果连接了多个设备，则递增*devIndex*将依次获取每个连接设备的序列号。

1. 获取当前所有设备的设备描述 连接的

字符 \*BufPtrs[3]; // 指向 3 数组的指针 指针

字符 缓冲区1[64]； // 描述缓冲区 第一的 设备 字符 缓冲区2[64]； // 用于描述第二个的缓冲区 设备

// 初始化指针数组 BufPtrs[0] = Buffer1;

BufPtrs[1] = 缓冲区2；

缓冲区指针[2] = 无效的; // 最后一个条目应该是 无效的

ftStatus = FT\_ListDevices(BufPtrs,&numDevs,FT\_LIST\_ALL|FT\_OPEN\_BY\_DESCRIPTION);如果（ftStatus == FT\_OK）{

// FT\_ListDevices OK，产品描述在Buffer1和Buffer2中， 和

// numDevs 包含连接的设备数量

}

别的 {

}

// FT\_ListDevices 失败

请注意，此示例假设连接了两个设备。如果连接更多设备，则必须增加指针数组的大小并分配更多描述缓冲区。

1. 获取当前所有设备的位置 连接的

长locIdBuf[16]；

ftStatus = FT\_ListDevices(locIdBuf,&numDevs,FT\_LIST\_ALL|FT\_OPEN\_BY\_LOCATION);如果（ftStatus == FT\_OK）{

// FT\_ListDevices OK，位置 ID 位于 locIdBuf 中，并且

// numDevs 包含连接的设备数量

}

别的 {

}

// FT\_ListDevices 失败

请注意，此示例假设连接的设备不超过 16 个。如果连接更多设备，则必须增加指针数组的大小。

## FT\_打开

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本）

Windows CE（4.2 及更高版本）

### 概括

打开设备并返回一个句柄，该句柄将用于后续访问。

### 定义

FT\_STATUS **FT\_Open** (int *iDevice* , FT\_HANDLE *\*ftHandle* )

### 参数

*iDevice* 要打开的设备的索引。指数为0 基于。

*ftHandle* 指向 FT\_HANDLE 类型变量的指针，其中句柄 将要 被存储。该句柄必须用于访问 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

虽然此函数可以通过将*iDevice设置*为 0、1、2 等来打开多个设备，但无法打开特定设备。要打开命名设备，请使用函数[FT\_OpenEx](#_bookmark11) 。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0,&ftHandle);如果（ftStatus == FT\_OK）{

// FT\_Open OK，使用ftHandle访问设备

}

别的 {

}

// FT\_Open 失败

## FT\_OpenEx

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

打开指定的设备并返回一个句柄，该句柄将用于后续访问。该设备可以

通过其序列号、设备描述或位置来指定。

此功能还可用于同时打开多个设备。可以通过序列号、设备描述或位置 ID（从 USB 设备的物理位置导出的位置信息）来指定多个设备。可以使用 USBView 实用程序获取特定 USB 端口的位置 ID，并以十六进制格式给出。连接到系统的设备的位置 ID 可以通过使用适当的标志调用[FT\_GetDeviceInfoList](#_bookmark7)或[FT\_ListDevices来获取。](#_bookmark9)

### 定义

FT\_STATUS **FT\_OpenEx** （PVOID *pvArg1* ，DWORD *dwFlags* ，FT\_HANDLE *\*ftHandle* ）

### 参数

*pvArg1* 指向参数的指针，其类型取决于 价值 dwFlags。它通常被解释为指向空终止的指针 细绳。

*dwFlags* [FT\_OPEN\_BY\_SERIAL\_NUMBER](#_bookmark93) , [FT\_OPEN\_BY\_DESCRIPTION](#_bookmark93) 或[FT\_OPEN\_BY\_LOCATION](#_bookmark93) 。

*ftHandle* 指向 FT\_HANDLE 类型变量的指针，其中句柄 将要 被存储。该句柄必须用于访问 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

*pvArg1*中指定的参数取决于*dwFlags* ：如果*dwFlags*为[*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93) ，则*pvArg1*被解释为指向表示设备序列号的以 null 结尾的字符串的指针；如果*dwFlags*为[*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) ，则*pvArg1*被解释为指向表示设备描述的以 null 结尾的字符串的指针；如果*dwFlags*为[*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) ，*则 pvArg1*被解释为包含设备位置 ID 的长值。请注意，Windows CE 和 Linux 不支持位置 ID。

*ftHandle是指向*要存储句柄的*FT\_HANDLE*类型变量的指针。必须使用此句柄来访问设备。

### 例子

下面的示例使用这些变量。

FT\_STATUS ft状态； FT\_STATUS ftStatus2; FT\_HANDLE ftHandle1; FT\_HANDLE ftHandle2;长 dwLoc；

1. 打开带有序列号的设备 “FT000001”

ftStatus = FT\_OpenEx((PVOID) "FT000001",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle1);如果（ftStatus == FT\_OK）{

// 成功 - 序列号为“FT000001”的设备已打开

}

别的 {

}

// 失败

1. 打开设备描述为“USB Serial”的设备 转换器”

ftStatus = FT\_OpenEx((PVOID) "USB 串口转换器",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle1);如果（ftStatus == FT\_OK）{

// 成功 - 设备描述为“USB Serial Converter”的设备已打开

}

别的 {

}

// 失败

1. 打开 2 个序列号为“FT000001”的设备并 “FT999999”

ftStatus = FT\_OpenEx((PVOID) "FT000001",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle1); ftStatus2 = FT\_OpenEx((PVOID) "FT999999",FT\_OPEN\_BY\_SERIAL\_NUMBER,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// 成功 - 两个设备都打开

}

别的 {

}

// 失败 - 一个或两个设备尚未打开

1. 打开 2 个描述为“USB Serial Converter”和“USB Pump”的设备 控制器”

ftStatus = FT\_OpenEx((PVOID) "USB 串口转换器",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle1); ftStatus2 = FT\_OpenEx((PVOID) "USB 泵控制器",FT\_OPEN\_BY\_DESCRIPTION,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// 成功 - 两个设备都打开

}

别的 {

}

// 失败 - 一个或两个设备尚未打开

1. 在某个位置打开设备 23

dwLoc = 0x23;

ftStatus = FT\_OpenEx((PVOID)dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle1);如果（ftStatus == FT\_OK）{

// 成功 - 位置 23 的设备已打开

}

别的 {

}

// 失败

1. 在位置 23 打开 2 个设备并 31

dwLoc = 0x23;

ftStatus = FT\_OpenEx((PVOID) dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle1); dwLoc = 0x31;

ftStatus2 = FT\_OpenEx((PVOID) dwLoc,FT\_OPEN\_BY\_LOCATION,&ftHandle2); if (ftStatus == FT\_OK && ftStatus2 == FT\_OK) {

// 成功 - 两个设备都打开

}

别的 {

}

// 失败 - 一个或两个设备尚未打开

## FT\_关闭

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

关闭打开的设备。

### 定义

FT\_STATUS **FT\_Close** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0,&ftHandle);如果（ftStatus == FT\_OK）{

// FT\_Open OK，使用ftHandle访问设备

// 完成后，调用 FT\_Close FT\_Close(ftHandle);

}

别的 {

}

// FT\_Open 失败

## FT\_读

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

从设备读取数据。

### 定义

FT\_STATUS **FT\_Read** (FT\_HANDLE *ftHandle,* LPVOID *lpBuffer,* DWORD *dwBytesToRead,*

LPDWORD lpdwBytes(返回)

### 参数

*ftHandle* 的句柄 设备。

*lpBuffer* 指向接收数据的缓冲区的指针 设备。

*dwBytesToRead* 要读取的字节数 设备。

*lpdwBytesReturned* 指向接收数字的 DWORD 类型变量的指针 的

从设备读取的字节数。

### 返回值

如果成功则返回 FT\_OK，否则返回 FT\_IO\_ERROR。

### 评论

*lpdwBytesReturned*中读取的字节数。

*dwBytesToRead*字节读入缓冲区之前，此函数不会返回。接收队列中的字节数可以通过调用[FT\_GetStatus](#_bookmark31)或[FT\_GetQueueStatus来确定](#_bookmark26)，并将其作为*dwBytesToRead传递给 FT\_Read* ，以便该函数读取设备并立即返回。

[FT\_SetTimeouts 的](#_bookmark18)调用中指定了读取超时值，则当计时器到期或已读取*dwBytesToRead （以先发生者为准）时，FT\_Read 返回。*如果发生超时， FT\_Read 将可用数据读入缓冲区并返回*FT\_OK* 。

在处理缓冲区时应使用函数返回值和*lpdwBytesReturned 。*如果返回值为*FT\_OK* ，并且*lpdwBytesReturned*等于*dwBytesToRead*则 FT\_Read 已正常完成。如果返回值为*FT\_OK* ，并且*lpdwBytesReturned*小于*dwBytesToRead* ，则发生超时并且读取已部分完成。请注意，如果发生超时并且没有读取数据，则返回值仍然是*FT\_OK* 。

返回值*FT\_IO\_ERROR*表明函数参数中存在错误，或者发生了 USB 断开连接等致命错误。

### 例子

1. 该示例展示了如何读取当前的所有数据 可用的。

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

DWORD 事件DWord；

DWORD TxBytes；

DWORD 接收字节；

已接收的 DWORD 字节数；字符 RxBuffer[256]；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (RxBytes > 0) {

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);如果（ftStatus == FT\_OK）{

// FT\_读取成功

}

别的 {

}

// FT\_读取失败

}

FT\_Close(ftHandle);

1. 此示例展示了如何在超时为 5 的情况下进行读取 秒。

FT\_HANDLE ftHandle; FT\_STATUS ft状态； DWORD 接收字节 = 10；已接收的 DWORD 字节数；字符 RxBuffer[256]；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

FT\_SetTimeouts(ftHandle,5000,0);

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);如果（ftStatus == FT\_OK）{

if (BytesReceived == RxBytes) {

// FT\_读取成功

}

别的 {

}

}

// FT\_读取超时

别的 {

}

// FT\_读取失败

FT\_Close(ftHandle);

## FT\_写

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将数据写入设备。

### 定义

FT\_STATUS **FT\_Write** (FT\_HANDLE *ftHandle,* LPVOID *lpBuffer,* DWORD *dwBytesToWrite,*

LPDWORD lpdwBytesWritten)

### 参数

*ftHandle* 的句柄 设备。

*lpBuffer* 指向包含要写入的数据的缓冲区的指针 到 装置。

*dwBytesToWrite* 要写入的字节数 设备。

*lpdwBytesWritten* 指向接收数字的 DWORD 类型变量的指针 的

写入设备的字节数。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle; FT\_STATUS ft状态； DWORD 写入字节数；

字符 TxBuffer[256]； // 包含要写入设备的数据

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_Write(ftHandle, TxBuffer, sizeof(TxBuffer), &BytesWritten);如果（ftStatus == FT\_OK）{

// FT\_写入成功

}

别的 {

}

// FT\_写入失败

FT\_Close(ftHandle);

## FT\_设置波特率

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置设备的波特率。

### 定义

FT\_STATUS **FT\_SetBaudRate** (FT\_HANDLE *ftHandle,* DWORD *dwBaudRate* )

### 参数

*ftHandle* 的句柄 设备。

*dwBaudRate* 波特率 速度。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetBaudRate(ftHandle, 115200); // 将波特率设置为 115200 if (ftStatus == FT\_OK) {

// FT\_SetBaudRate 确定

}

别的 {

}

// FT\_SetBaudRate 失败

FT\_Close(ftHandle);

## FT\_设置除数

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置设备的波特率。用于设置非标准波特率。

### 定义

FT\_STATUS **FT\_SetDivisor** (FT\_HANDLE *ftHandle,* USHORT *usDivisor* )

### 参数

*ftHandle* 的句柄 设备。

*usDivisor* 除数。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

不再需要此函数，因为[FT\_SetBaudRate](#_bookmark15)现在将自动计算请求的波特率所需的除数。应用笔记“设置 FT8U232AM 的波特率”可从 FTDI 网站的应用笔记部分获取，介绍了如何计算非标准波特率的除数。

## FT\_SetDataCharacteristics

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

该函数设置设备的数据特征。

### 定义

FT\_STATUS **FT\_SetDataCharacteristics** (FT\_HANDLE *ftHandle,* UCHAR *uWordLength,*

UCHAR uStopBits、UCHAR uParity)

### 参数

*ftHandle* 的句柄 设备。

*uWordLength* 每个字的位数 - 必须是[FT\_BITS\_8](#_bookmark94)或 [FT\_BITS\_7](#_bookmark94) 。

*uStopBits* 停止位的数量 - 必须是 [FT\_STOP\_BITS\_1](#_bookmark95) 或[FT\_STOP\_BITS\_2](#_bookmark95) 。

*uParity* 奇偶校验 - 必须是 [FT\_PARITY\_NONE](#_bookmark96) , [FT\_PARITY\_ODD](#_bookmark96) 、 [FT\_PARITY\_EVEN](#_bookmark96) 、 [FT\_PARITY\_MARK](#_bookmark96)或[FT\_PARITY 空间](#_bookmark96)。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

// 设置8个数据位，1个停止位，无奇偶校验

ftStatus = FT\_SetDataCharacteristics(ftHandle, FT\_BITS\_8, FT\_STOP\_BITS\_1, FT\_PARITY\_NONE);

如果（ftStatus == FT\_OK）{

// FT\_SetDataCharacteristics 确定

}

别的 {

}

// FT\_SetDataCharacteristics 失败

FT\_Close(ftHandle);

## FT\_设置超时

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置设备的读取和写入超时。

### 定义

FT\_STATUS **FT\_SetTimeouts** (FT\_HANDLE *ftHandle,* DWORD *dwReadTimeout,*

DWORD dwWriteTimeout)

### 参数

*ftHandle* 的句柄 设备。

*dwReadTimeout* 读取超时时间 毫秒。

*dwWriteTimeout* 写入超时 毫秒。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

// 设置读取超时为 5 秒，写入超时为 1 秒 ftStatus = FT\_SetTimeouts(ftHandle, 5000, 1000); if (ftStatus == FT\_OK) {

// FT\_SetTimeouts 确定

}

别的 {

}

// FT\_SetTimeouts 失败

FT\_Close(ftHandle);

## FT\_设置流量控制

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置设备的流量控制。

### 定义

FT\_STATUS **FT\_SetFlowControl** (FT\_HANDLE *ftHandle,* USHORT *usFlowControl,*

UCHAR *uXon、* UCHAR *uXoff* )

### 参数

*ftHandle* 的句柄 设备。

*usFlowControl* 必须是其中之一 [FT\_FLOW\_NONE](#_bookmark97) , [FT\_FLOW\_RTS\_CTS](#_bookmark97) 、 [FT\_FLOW\_DTR\_DSR](#_bookmark97)或 [FT\_FLOW\_XON\_XOFF](#_bookmark97) 。

*uXon* 用于表示 Xon 的字符。仅在流量时使用 控制 是[FT\_FLOW\_XON\_XOFF](#_bookmark97) 。

*uXoff* 用于表示 Xoff 的字符。仅在流量时使用 控制 是[FT\_FLOW\_XON\_XOFF](#_bookmark97) 。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

// 设置RTS/CTS流控

ftStatus = FT\_SetFlowControl(ftHandle, FT\_FLOW\_RTS\_CTS, 0x11, 0x13);如果（ftStatus == FT\_OK）{

// FT\_SetFlowControl 确定

}

别的 {

}

// FT\_SetFlowControl 失败

FT\_Close(ftHandle);

## FT\_设置数据

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置数据终端就绪（DTR）控制信号。

### 定义

FT\_STATUS **FT\_SetDtr** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数断言设备的数据终端就绪 (DTR) 线。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetDtr(ftHandle);如果（ftStatus == FT\_OK）{

// FT\_SetDtr 确定

}

别的 {

}

// FT\_SetDtr 失败

FT\_Close(ftHandle);

## FT\_ClrDtr

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

该函数清除数据终端就绪（DTR）控制信号。

### 定义

FT\_STATUS **FT\_ClrDtr** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数取消断言设备的数据终端就绪 (DTR) 线。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_ClrDtr(ftHandle);如果（ftStatus == FT\_OK）{

// FT\_ClrDtr 确定

}

别的 {

}

// FT\_ClrDtr 失败

FT\_Close(ftHandle);

## FT\_设置Rts

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置请求发送（RTS）控制信号。

### 定义

FT\_STATUS **FT\_SetRts** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数断言设备的请求发送 (RTS) 线。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetRts(ftHandle);如果（ftStatus == FT\_OK）{

// FT\_SetRts 确定

}

别的 {

}

// FT\_SetRts 失败

FT\_Close(ftHandle);

## FT\_ClrRts

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数清除请求发送（RTS）控制信号。

### 定义

FT\_STATUS **FT\_ClrRts** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数取消声明设备的请求发送 (RTS) 线路。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_ClrRts(ftHandle);如果（ftStatus == FT\_OK）{

// FT\_ClrRts 确定

}

别的 {

}

// FT\_ClrRts 失败

FT\_Close(ftHandle);

## FT\_GetMo demStatus

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

从设备获取调制解调器状态和线路状态。

### 定义

FT\_STATUS **FT\_GetModemStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwModemStatus* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwModemStatus* 指向 DWORD 类型变量的指针，该变量接收 调制解调器

来自设备的状态和线路状态。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

*lpdwModemStatus值*的最低有效字节保存调制解调器状态。在 Windows 和 Windows CE 上，线路状态保存在*lpdwModemStatus*值的第二个最低有效字节中。

调制解调器状态按如下位映射：清除发送 ( [CTS](#_bookmark100) ) = 0x10、数据集就绪 ( [DSR](#_bookmark100) ) = 0x20、振铃指示器 ( [RI](#_bookmark100) ) = 0x40、数据载波检测 ( [DCD](#_bookmark100) ) = 0x80。

线路状态按如下位映射：溢出错误 ( [OE](#_bookmark101) ) = 0x02、奇偶校验错误 ( [PE](#_bookmark101) ) = 0x04、帧错误 ( [FE](#_bookmark101) ) = 0x08 、中断中断 ( [BI](#_bookmark101) ) = 0x10。

### 例子

FT\_HANDLE ftHandle； FT\_STATUS ft状态； DWORD dwModemStatus = 0; DWORD dwLineStatus = 0;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_GetModemStatus(ftHandle, &dwModemStatus);如果（ftStatus == FT\_OK）{

// FT\_GetModemStatus 确定

// 线路状态是 dwModemStatus 值的第二个字节 dwLineStatus = ((dwModemStatus >> 8) & 0x000000FF);

// 现在屏蔽调制解调器状态字节 dwModemStatus = (dwModemStatus & 0x000000FF);

}

别的 {

}

// FT\_GetModemStatus 失败

FT\_Close(ftHandle);

## FT\_GetQueueStatus

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

获取接收队列中的字节数。

### 定义

FT\_STATUS **FT\_GetQueueStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwAmountInRxQueue* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwAmountInRxQueue* 指向接收数字的 DWORD 类型变量的指针 的

接收队列中的字节。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle； FT\_STATUS ft状态； DWORD 接收字节；

已接收的 DWORD 字节数；字符 RxBuffer[256]；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

FT\_GetQueueStatus(ftHandle,&RxBytes); if (RxBytes > 0) {

ftStatus = FT\_Read(ftHandle,RxBuffer,RxBytes,&BytesReceived);如果（ftStatus == FT\_OK）{

// FT\_读取成功

}

别的 {

}

}

// FT\_读取失败

FT\_Close(ftHandle);

## FT\_获取设备信息

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取打开设备的设备信息。

### 定义

FT\_STATUS **FT\_GetDeviceInfo** (FT\_HANDLE *ftHandle,* FT\_DEVICE *\*pftType,*

LPDWORD *lpdwID* 、PCHAR pcSerialNumber、PCHAR pcDescription、PVOID pvDummy)

### 参数

*ftHandle* 的句柄 设备。

*pftType* 指向 unsigned long 的指针以存储设备 类型。

*lpdwID* 指向 unsigned long 的指针以存储设备 ID。

*pcSerialNumber* 指向缓冲区的指针，用于将设备序列号存储为 无效的-

终止的字符串。

*pcDescription* 指向缓冲区的指针，用于存储设备描述 A 以 null 结尾的字符串。

*pvDummy* 保留供将来使用 - 应设置为 无效的。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数用于返回设备类型、设备ID、设备描述和序列号。

设备 ID 以 DWORD 形式编码 - 最高有效字包含供应商 ID，最低有效字包含产品 ID。所以返回的ID 0x04036001对应的是设备ID VID\_0403&PID\_6001。

### 例子

FT\_HANDLE ftHandle; FT\_DEVICE ftDevice； FT\_STATUS ft状态； DWORD 设备ID；

字符序列号[16]；字符描述[64]；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_GetDeviceInfo(

ftHandle、&ftDevice、&deviceID、序列号、描述、NULL

）；

如果（ftStatus == FT\_OK）{

if (ftDevice == FT\_DEVICE\_232H)

; // 设备为FT232H

否则如果（ftDevice == FT\_DEVICE\_4232H）

; //设备是FT4232H

否则如果（ftDevice == FT\_DEVICE\_2232H）

; // 设备为FT2232H

否则如果（ftDevice == FT\_DEVICE\_232R）

; // 设备是FT232R

否则如果（ftDevice == FT\_DEVICE\_2232C）

; // 设备是 FT2232C/L/D else if (ftDevice == FT\_DEVICE\_BM)

; // 设备是 FTU232BM else if (ftDevice == FT\_DEVICE\_AM)

; // 设备是 FT8U232AM

别的

; // 未知设备（这不应该发生！）

}

别的 {

}

// deviceID 包含编码后的设备 ID

// SerialNumber、Description 包含以 0 结尾的字符串

// FT\_GetDeviceType 失败！

FT\_Close(ftHandle);

## FT\_GetDriverVersion

### 支持的操作系统

Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数返回D2XX驱动程序版本号。

### 定义

FT\_STATUS **FT\_GetDriverVersion** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwDriverVersion* )

### 参数

*ftHandle* 句柄 这 设备。 *lpdwDriverVersion* 指向驱动程序的指针 版本 数字。**返回 价值**

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

版本号由 4 字节字段（无符号长整型）中包含的主版本号、次版本号和内部版本号组成。 Byte0（最低有效）保存构建版本，Byte1 保存次要版本，Byte2 保存主要版本。 Byte3 当前设置为零。

例如，驱动程序版本“2.04.06”表示为 0x00020406。请注意，在调用此函数之前必须打开设备。

### 例子

FT\_HANDLE ftHandle; FT\_STATUS ft状态； DWORD dwDriverVer;

// 获取驱动版本

ftStatus = FT\_Open(0,&ftHandle);如果（ftStatus == FT\_OK）{

ftStatus = FT\_GetDriverVersion(ftHandle,&dwDriverVer); if (ftStatus == FT\_OK)

printf("驱动程序版本 = 0x%x\n",dwDriverVer);

别的

printf("读取驱动程序版本时出错\n");

FT\_Close(ftHandle);

}

## FT\_GetLibraryVersion

### 支持的操作系统

Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数返回 D2XX DLL 版本号。

### 定义

FT\_STATUS **FT\_GetLibraryVersion** (LPDWORD *lpdwDLLVersion* )

### 参数

*lpdwDLLVersion* 指向 DLL 版本的指针 数字。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

版本号由 4 字节字段（无符号长整型）中包含的主版本号、次版本号和内部版本号组成。 Byte0（最低有效）保存构建版本，Byte1 保存次要版本，Byte2 保存主要版本。 Byte3 当前设置为零。

例如，D2XX DLL 版本“3.01.15”表示为 0x00030115。请注意，该函数不使用句柄，因此无需打开设备即可调用它。

### 例子

FT\_STATUS ft状态；

DWORD dwLibraryVer;

// 获取DLL版本

ftStatus = FT\_GetLibraryVersion(&dwLibraryVer); if (ftStatus == FT\_OK)

printf("库版本 = 0x%x\n",dwLibraryVer);

别的

printf("读取库版本时出错\n");

## FT\_GetComPortNumber

**支持的操作系统**Windows（2000 及更高版本）**摘要**

检索与设备关联的 COM 端口。

### 定义

FT\_STATUS **FT\_GetComPortNumber** (FT\_HANDLE *ftHandle,* LPLONG *lplComPortNumber* )

### 参数

*ftHandle* 的句柄 设备。

*lplComPortNumber* 指向接收 COM 的 LONG 类型变量的指针 港口 与相关的号码 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此功能仅在使用 Windows CDM 驱动程序时可用，因为 D2XX 和 VCP 驱动程序可以同时安装。

如果没有与设备关联的 COM 端口， *lplComPortNumber*的值为 -1。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); LONG lComPortNumber；

ftStatus = FT\_GetComPortNumber(ftHandle,&lComPortNumber);如果（ftStatus == FT\_OK）{

if (lComPortNumber == -1) {

// 未分配 COM 端口

}

别的 {

}

别的 {

}

// COM 端口分配有 lComPortNumber 中保存的编号

// FT\_GetComPortNumber 失败！

}

FT\_Close(ftHandle);

## FT\_获取状态

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取设备状态，包括接收队列中的字符数、接收队列中的字符数

传输队列和当前事件状态。

### 定义

FT\_STATUS **FT\_GetStatus** (FT\_HANDLE *ftHandle,* LPDWORD *lpdwAmountInRxQueue* ,

LPDWORD lpdwAmountInTxQueue、LPDWORD lpdwEventStatus)

### 参数

*ftHandle* 的句柄 设备。

*lpdwAmountInRxQueue*指向 DWORD 类型变量的指针，该变量接收接收队列中的字符数。

*lpdwAmountInTxQueue*指向 DWORD 类型变量的指针，该变量接收传输队列中的字符数。

*lpdwEventStatus* 指向接收当前事件的 DWORD 类型变量的指针 状态 事件的 地位。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

有关如何使用此函数的示例，请参阅[FT\_SetEventNotification中的示例代码](#_bookmark32)。

## FT\_设置事件通知

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

设置事件通知的条件。

### 定义

FT\_STATUS **FT\_SetEventNotification** (FT\_HANDLE *ftHandle,* DWORD *dwEventMask,*

PVOID *（PVArg* ）

### 参数

*ftHandle* 的句柄 设备。

*dwEventMask* 导致事件发生的条件 放。

*pvArg* 解释为一个句柄 事件。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

应用程序可以使用此函数来设置允许线程阻塞直到满足其中一个条件的条件。通常，应用程序将创建一个事件，调用此函数，然后阻止该事件。当条件满足时，事件被设置，并且应用程序线程被解除阻塞。

*dwEventMask*是一个位图，描述应用程序感兴趣的事件。pvArg*被*解释为应用程序创建的事件的句柄。如果满足事件条件之一，则设置该事件。

如果在*dwEventMask*中设置了[*FT\_EVENT\_RXCHAR*](#_bookmark99) ，则当设备接收到字符时将设置该事件。

如果在*dwEventMask*中设置了[*FT\_EVENT\_MODEM\_STATUS*](#_bookmark99) ，则当设备检测到调制解调器信号发生变化时，将设置该事件。

如果在*dwEventMask*中设置了[*FT\_EVENT\_LINE\_STATUS*](#_bookmark99) ，则当设备检测到线路状态发生变化时，将设置该事件。

### 例子

1. 此示例对 Windows 和 Windows CE 有效，并显示如何等待接收字符或调制解调器更改 地位。

// 首先，创建事件并调用 FT\_SetEventNotification。 FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);处理 hEvent；

DWORD 事件掩码；

hEvent = 创建事件(

无效的，

false, // 自动重置事件 false, // 无信号状态 (LPCWSTR) ""

）；

事件掩码 = FT\_EVENT\_RXCHAR | FT\_EVENT\_MODEM\_STATUS；

ftStatus = FT\_SetEventNotification(ftHandle,EventMask,hEvent);

// 稍后，通过等待事件来阻塞应用程序线程，然后当事件发生时

// 发生后，确定导致该事件的条件，并进行相应的处理。 WaitForSingleObject(hEvent,INFINITE);

DWORD 事件DWord；

DWORD 接收字节；

DWORD TxBytes；

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (EventDWord & FT\_EVENT\_MODEM\_STATUS) {

// 检测到调制解调器状态事件，因此获取当前调制解调器状态 FT\_GetModemStatus(ftHandle,&Status);

if (状态 & 0x00000010) {

// CTS 为高

}

别的 {

}

// CTS 低

if (状态 & 0x00000020) {

// DSR 为高

}

别的 {

}

}

// DSR 低

if (RxBytes > 0) {

// 调用 FT\_Read() 从设备获取接收到的数据

}

1. 此示例对 Linux 有效，并显示如何等待接收字符或调制解调器发生变化 地位。

// 首先，创建事件并调用 FT\_SetEventNotification。 FT\_HANDLE ftHandle；

FT\_STATUS ft状态；

EVENT\_HANDLE 呃；

DWORD 事件掩码；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

pthread\_mutex\_init(&eh.eMutex, NULL); pthread\_cond\_init(&eh.eCondVar, NULL);

事件掩码 = FT\_EVENT\_RXCHAR | FT\_EVENT\_MODEM\_STATUS；

ftStatus = FT\_SetEventNotification(ftHandle, EventMask, (PVOID)&eh);

// 稍后，通过等待事件来阻塞应用程序线程，然后当事件发生时

// 发生后，判断引发该事件的条件，并进行处理 因此。

pthread\_mutex\_lock(&eh.eMutex); pthread\_cond\_wait(&eh.eCondVar, &eh.eMutex); pthread\_mutex\_unlock(&eh.eMutex);

DWORD 事件DWord；

DWORD 接收字节；

DWORD TxBytes；

双字状态；

FT\_GetStatus(ftHandle,&RxBytes,&TxBytes,&EventDWord); if (EventDWord & FT\_EVENT\_MODEM\_STATUS) {

// 检测到调制解调器状态事件，因此获取当前调制解调器状态 FT\_GetModemStatus(ftHandle,&Status);

if (状态 & 0x00000010) {

// CTS 为高

}

别的 {

}

// CTS 低

if (状态 & 0x00000020) {

// DSR 为高

}

别的 {

}

}

// DSR 低

if (RxBytes > 0) {

// 调用 FT\_Read() 从设备获取接收到的数据

}

FT\_Close(ftHandle);

## FT\_设置字符

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

该函数设置设备的特殊字符。

### 定义

FT\_STATUS **FT\_SetChars** (FT\_HANDLE *ftHandle,* UCHAR *uEventCh,* UCHAR *uEventChEn,*

UCHAR uErrorCh、UCHAR uErrorChEn)

### 参数

*ftHandle* 的句柄 设备。

*uEventCh* 事件 特点。

*uEventChEn* 如果事件字符被禁用则为 0，非零 否则。

*uErrorCh* 错误 特点。

*uErrorChEn* 如果禁用错误字符则为 0，非零 否则。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数允许在数据流中插入指定的字符来表示事件触发或发生的错误。

## FT\_SetBreakOn

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

设置设备的 BREAK 条件。

### 定义

FT\_STATUS **FT\_SetBreakOn** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_SetBreakOn(ftHandle);

如果（ftStatus == FT\_OK）{

// FT\_SetBreakOn 确定

}

别的 {

}

// FT\_SetBreakOn 失败

FT\_Close(ftHandle);

## FT\_SetBreakOff

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

重置设备的 BREAK 条件。

### 定义

FT\_STATUS **FT\_SetBreakOff** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_SetBreakOff(ftHandle);

如果（ftStatus == FT\_OK）{

// FT\_SetBreakOff 确定

}

别的 {

}

// FT\_SetBreakOff 失败

FT\_Close(ftHandle);

## FT\_清除

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

此函数清除设备中的接收和发送缓冲区。

### 定义

FT\_STATUS **FT\_Purge** (FT\_HANDLE *ftHandle,* DWORD *dwMask* )

### 参数

*ftHandle* 的句柄 设备。

*uEventCh* [FT\_PURGE\_RX](#_bookmark98)和的组合 [FT\_PURGE\_TX](#_bookmark98) 。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_Purge(ftHandle, FT\_PURGE\_RX | FT\_PURGE\_TX); // 清除 Rx 和 Tx 缓冲区 if (ftStatus == FT\_OK) {

// FT\_Purge 确定

}

别的 {

}

// FT\_Purge 失败

FT\_Close(ftHandle);

## FT\_重置设备

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数向设备发送重置命令。

### 定义

FT\_STATUS **FT\_ResetDevice** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); ftStatus = FT\_ResetDevice(ftHandle);

如果（ftStatus == FT\_OK）{

// FT\_ResetDevice 确定

}

别的 {

}

// FT\_ResetDevice 失败

FT\_Close(ftHandle);

## FT\_重置端口

**支持的操作系统**Windows（2000 及更高版本）**摘要**

向端口发送重置命令。

### 定义

FT\_STATUS **FT\_ResetPort** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该功能用于在发生故障后尝试恢复端口。它不等同于拔出-重新插入事件。对于相当于拔出-重新插入事件，请使用[FT\_CyclePort](#_bookmark39) 。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_ResetPort(ftHandle);如果（ftStatus == FT\_OK）{

// 端口已重置

}

别的 {

}

// FT\_ResetPort 失败！

## FT\_CyclePort

**支持的操作系统**Windows（2000 及更高版本）**摘要**

向 USB 端口发送循环命令。

### 定义

FT\_STATUS **FT\_CyclePort** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此功能的效果与从 USB 断开然后重新连接设备的效果相同。可能使用此功能的情况是发生了致命错误，并且在不拔出并重新插入 USB 电缆的情况下很难或不可能恢复。该功能也可以在重新编程 EEPROM 后使用，以强制 FTDI 设备读取新的 EEPROM 内容，否则需要物理断开重新连接。

由于重新加载驱动程序时当前会话不会恢复，因此应用程序必须能够在调用此函数后恢复。应用程序有责任在成功调用 FT\_CyclePort 后关闭句柄。

对于 FT4232H、FT2232H 和 FT2232 设备，FT\_CyclePort 仅在 Windows XP 及更高版本下工作。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

ftStatus = FT\_CyclePort(ftHandle);如果（ftStatus == FT\_OK）{

// 端口已被循环。

// 关闭句柄。

ftStatus = FT\_Close(ftHandle);

}

别的 {

}

// FT\_CyclePort 失败！

## FT\_重新扫描

**支持的操作系统**Windows（2000 及更高版本）**摘要**

当尝试以编程方式恢复设备时，此函数很有用。

### 定义

FT\_STATUS **FT\_重新扫描**()

### 参数

没有任何

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

调用 FT\_Rescan 相当于单击设备管理器中的“扫描硬件更改”按钮。仅检查 USB 硬件是否有新设备。所有 USB 设备都会被扫描，而不仅仅是 FTDI 设备。

### 例子

FT\_STATUS ft状态；

ftStatus = FT\_Rescan(); if(ftStatus!= FT\_OK) {

// FT\_Rescan 失败！返回;

}

## FT\_重新加载

**支持的操作系统**Windows（2000 及更高版本）**摘要**

此函数强制重新加载具有特定 VID 和 PID 组合的设备的驱动程序。

### 定义

FT\_STATUS **FT\_Reload** （字*wVID* ，字*wPID* ）

### 参数

*wVID* 要重新加载驱动程序的设备的供应商 ID 为了。

*wPID* 要重新加载驱动程序的设备的产品 ID 为了。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

调用 FT\_Reload 会强制操作系统卸载并重新加载指定设备 ID 的驱动程序。如果VID和PID参数为空，USB根集线器的驱动程序将被重新加载，导致所有连接的USB设备重新加载其驱动程序。请注意，当从 32 位应用程序调用时，此函数将无法在 64 位 Windows 上正常工作。

### 例子

1. 此示例显示如何调用 FT\_Reload 来重新加载标准 FT232R 设备（VID 0x0403，PID 0x6001）。

FT\_STATUS ft状态；字wVID = 0x0403；字 wPID = 0x6001；

ftStatus = FT\_Reload(wVID,wPID); if(ftStatus!= FT\_OK) {

// FT\_Reload 失败！返回;

}

1. 此示例演示如何调用 FT\_Reload 重新加载所有 USB 设备的驱动程序。 FT\_状态 ft状态；

字wVID = 0x0000；字 wPID = 0x0000；

ftStatus = FT\_Reload(wVID,wPID); if(ftStatus!= FT\_OK) {

// FT\_Reload 失败！返回;

}

## FT\_SetResetPipeRetryCount

### 支持的操作系统

Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

设置 ResetPipeRetryCount 值。

### 定义

FT\_STATUS **FT\_SetResetPipeRetryCount** (FT\_HANDLE *ftHandle,* DWORD *dwCount* )

### 参数

*ftHandle* 的句柄 设备。

*dwCount* Unsigned long 包含必需的 重置管道重试计数。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数用于设置ResetPipeRetryCount。 ResetPipeRetryCount 控制驱动程序尝试重置发生错误的管道的最大次数。

ResetPipeRequestRetryCount 默认为 50。在发生大量 USB 错误的嘈杂环境中，可能需要增加此值。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwRetryCount；

dw重试计数 = 100;

ftStatus = FT\_SetResetPipeRetryCount(ftHandle,dwRetryCount);如果（ftStatus == FT\_OK）{

// ResetPipeRetryCount 设置为 100

}

别的 {

}

// FT\_SetResetPipeRetryCount 失败！

## FT\_StopInTask

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

停止驱动程序的 IN 任务。

### 定义

FT\_STATUS **FT\_StopInTask** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数用于将驱动程序的 IN 任务（读）置于等待状态。它可用于连续接收数据的情况，以便在不接收更多数据的情况下清除设备。它与[FT\_RestartInTask一起使用](#_bookmark44)，设置 IN 任务再次运行。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

做 {

ftStatus = FT\_StopInTask(ftHandle);

while (ftStatus!= FT\_OK);

//

// 做一些事情 - 例如清除设备

//

做 {

ftStatus = FT\_RestartInTask(ftHandle);

while (ftStatus!= FT\_OK); FT\_Close(ftHandle);

## FT\_RestartInTask

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

重新启动驱动程序的 IN 任务。

### 定义

FT\_STATUS **FT\_RestartInTask** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数用于在通过调用[FT\_StopInTask停止驱动程序的 IN 任务（读取）后重新启动该任务](#_bookmark43)。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

做 {

ftStatus = FT\_StopInTask(ftHandle);

while (ftStatus!= FT\_OK);

//

// 做一些事情 - 例如清除设备

//

做 {

ftStatus = FT\_RestartInTask(ftHandle);

while (ftStatus!= FT\_OK); FT\_Close(ftHandle);

## FT\_SetDeadman超时

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

此函数允许 USB 请求保持未完成状态的最长时间（以毫秒为单位）

放。

### 定义

FT\_STATUS **FT\_SetDeadmanTimeout** (FT\_HANDLE *ftHandle* , DWORD *dwDeadmanTimeout* )

### 参数

*ftHandle* 的句柄 设备。

*dwDeadmanTimeout* Deadman 超时值（以毫秒为单位）。默认值为 5000。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

Deadman 超时在 FTDI 网站的应用笔记 AN232B-10 高级驱动程序选项中称为 USB 超时。大多数用户不太可能需要此功能。

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ft状态；

双字 dwDeadmanTimeout = 6000;

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetDeadmanTimeout(ftHandle,dwDeadmanTimeout);如果（ftStatus == FT\_OK）{

// 将 Deadman 计时器设置为 6 秒

}

别的 {

}

// FT\_SetDeadmanTimeout 失败！

FT\_Close(ftHandle);

## FT\_IoCtl

未记录的功能。

## FT\_SetWaitMask

未记录的功能。

## FT\_等待掩码

未记录的功能。

# EEPROM编程接口 功能

FTDI 器件 EEPROM 可以使用本节中列出的函数进行读取和编程。请注意以下信息：

* 制造商、制造商 ID、说明和序列号字符串的最大长度为 48 个字（1 个字 = 2 个字） 字节）。
* 序列号的前两个字符是制造商 ID。
* 制造商字符串长度加上描述字符串长度小于或等于 40 个字符，且满足以下条件 功能：
  + [FT\_EE\_读](#_bookmark53)
  + [FT\_EE\_程序](#_bookmark55)
  + [FT\_EE\_ProgramEx](#_bookmark56)
  + [FT\_EEPROM\_读取](#_bookmark60)
  + [FT\_EEPROM\_程序](#_bookmark61)
* 单端口设备（例如 FT232R、 FT-

X) 和多端口设备（例如 FT2232H、FT4232H）上的 14 个字符。如果它更长，那么它可能会被截断并且不会有空终止符。

例如，多端口设备上的 15 个字符长的序列号将具有 16 个字符长的有效序列号，因为序列号附加有通道标识符（A、B 等）。用于从 API 返回字符串的缓冲区大小仅为 16 个字符，因此 NULL 终止符将丢失。

如果 EEPROM 或设备配置中的序列号或描述太长，则 FT\_GetDeviceInfo 和 FT\_ListDevices 返回的字符串可能不会以 NULL 结尾

## FT\_ReadEE

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

从 EEPROM 位置读取一个值。

### 定义

FT\_STATUS **FT\_ReadEE** (FT\_HANDLE *ftHandle* , DWORD *dwWordOffset* , LPWORD *lpwValue* )

### 参数

*ftHandle* 的句柄 设备。

*dwWordOffset* 要读取的 EEPROM 位置 从。

*lpwValue* 指向从读取的 WORD 值的指针 EEPROM。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

FTDI 器件的 EEPROM 按 WORD 组织，因此返回的每个值都是 16 位宽。

## FT\_WriteEE

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将值写入 EEPROM 位置。

### 定义

FT\_STATUS **FT\_WriteEE** (FT\_HANDLE *ftHandle* , DWORD *dwWordOffset* , WORD *wValue* )

### 参数

*ftHandle* 的句柄 设备。

*dwWordOffset* 要读取的 EEPROM 位置 从。

*wValue* 写入的 WORD 值 EEPROM。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

FTDI 器件的 EEPROM 按 WORD 组织，因此写入 EEPROM 的每个值都是 16 位宽。

## FT\_EraseEE

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

擦除设备 EEPROM。

### 定义

FT\_STATUS **FT\_EraseEE** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将擦除 EEPROM 的全部内容，包括用户区域。请注意，FT232R 和 FT245R 器件具有无法擦除的内部 EEPROM。

## FT\_EE\_读

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

读取 EEPROM 的内容。

### 定义

FT\_STATUS **FT\_EE\_Read** (FT\_HANDLE *ftHandle* , PFT\_PROGRAM\_DATA *pData* )

### 参数

*ftHandle* 的句柄 设备。

*pData* 指向类型结构的指针 FT\_PROGRAM\_DATA。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*pData解释为指向FT\_PROGRAM\_DATA*类型结构的指针，该结构包含要从 EEPROM 读取的数据的存储空间。

该函数不对缓冲区大小执行任何检查，因此*FT\_PROGRAM\_DATA*结构中传递的缓冲区必须足够大以容纳它们各自的字符串（包括空终止符）。以下示例中显示的尺寸绰绰有余，如有必要可以向下舍入。限制是*Manufacturer*字符串长度加上*Description*字符串长度小于或等于40 人物。

请注意，必须告知 DLL正在使用哪个版本的*FT\_PROGRAM\_DATA结构。*这是通过结构的*Signature1* 、 *Signature2*和*Version元素完成的。 Signature1*应始终为*0x00000000* ， *Signature2*应始终为*0xFFFFFFFF* ，并且*Version*可以设置为使用所需的版本。为了与所有当前设备兼容，*版本*应设置为FTD2XX.h 中定义的*FT\_PROGRAM\_DATA结构的最新版本。*

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);如果（ftStatus！= FT\_OK）{

// FT\_打开失败！

}

FT\_PROGRAM\_DATA ftData；字符ManufacturerBuf[32]；字符ManufacturerIdBuf[16]；字符描述Buf[64]；字符 序列号Buf[16];

ftData.Signature1 = 0x00000000; ftData.Signature2 = 0xffffffff;

ftData.版本 = 0x00000005； // EEPROM 结构 FT232H 扩展名 ftData.Manufacturer = 制造商Buf；

ftData.ManufacturerId =ManufacturerIdBuf; ftData.Description = DescriptionBuf; ftData.SerialNumber = SerialNumberBuf;

ftStatus = FT\_EE\_Read(ftHandle, &ftData);如果（ftStatus == FT\_OK）{

// FT\_EE\_Read OK，数据在 ftData 中可用

}

别的 {

}

// FT\_EE\_读取失败！

FT\_Close(ftHandle);

## FT\_EE\_ReadEx

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

读取EEPROM的内容并分别传递字符串。

### 定义

FT\_STATUS **FT\_EE\_ReadEx** (FT\_HANDLE *ftHandle* , PFT\_PROGRAM\_DATA *pData* ,

char *\* 制造商*、 char \* 制造商 ID、 char \* 说明、 char \* 序列号）

### 参数

*ftHandle* 的句柄 设备。

*pData* 指向类型结构的指针 FT\_PROGRAM\_DATA。

*\*Manufacturer* 指向包含空终止字符串的指针 制造商

姓名。

*\*ManufacturerId* 指向包含制造商的空终止字符串的指针 ID。

*\*描述* 指向包含空终止字符串的指针 这 设备描述。

*\*SerialNumber* 指向包含设备的以 null 结尾的字符串的指针 连续剧

数字。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

标准[FT\_EE\_Read](#_bookmark53)函数的这种变体是为了提供对 LabVIEW 等语言的支持，当字符串指针包含在结构中时，可能会出现问题。

该函数将参数*pData解释为指向FT\_PROGRAM\_DATA*类型结构的指针，该结构包含要从 EEPROM 读取的数据的存储空间。

该函数不对缓冲区大小执行任何检查，因此*FT\_PROGRAM\_DATA*结构中传递的缓冲区必须足够大以容纳它们各自的字符串（包括空终止符）。

请注意，必须告知 DLL正在使用哪个版本的*FT\_PROGRAM\_DATA结构。*这是通过结构的*Signature1* 、 *Signature2*和*Version元素完成的。 Signature1*应始终为*0x00000000* ， *Signature2*应始终为*0xFFFFFFFF* ，并且*Version*可以设置为使用所需的版本。为了与所有当前设备兼容，*版本*应设置为FTD2XX.h 中定义的*FT\_PROGRAM\_DATA结构的最新版本。*

*FT\_PROGRAM\_DATA结构*中的字符串参数应作为 DWORD 传递，以避免参数重叠。所有字符串指针均与*FT\_PROGRAM\_DATA结构*分开传递。

## FT\_EE\_程序

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

对 EEPROM 进行编程。

### 定义

FT\_STATUS **FT\_EE\_Program** (FT\_HANDLE *ftHandle* , PFT\_PROGRAM\_DATA *pData* )

### 参数

*ftHandle* 的句柄 设备。

*pData* 指向类型结构的指针 FT\_PROGRAM\_DATA。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*pData解释为指向FT\_PROGRAM\_DATA*类型结构的指针，该结构包含要写入 EEPROM 的数据。数据被写入EEPROM，然后读回并验证。

如果*FT\_PROGRAM\_DATA中的SerialNumber*字段为 NULL，或者*SerialNumber*指向 NULL 字符串，则将生成基于*ManufacturerId和当前日期和时间的序列号。*制造*商*字符串长度加上*描述*字符串长度必须小于或等于 40 个字符。

请注意，必须告知 DLL正在使用哪个版本的*FT\_PROGRAM\_DATA结构。*这是通过结构的*Signature1* 、 *Signature2*和*Version元素完成的。 Signature1*应始终为*0x00000000* ， *Signature2*应始终为*0xFFFFFFFF* ，并且*Version*可以设置为使用所需的版本。为了与所有当前设备兼容，*版本*应设置为FTD2XX.h 中定义的*FT\_PROGRAM\_DATA结构的最新版本。*

如果*pData*为 NULL，则结构版本将默认为 0（原始 BM 系列），并且器件将使用默认数据进行编程：

### 例子

此示例显示如何对 FT232B 器件的 EEPROM 进行编程。需要为其他设备类型设置其他参数。

// 用于对 BM 设备进行编程的版本 4 结构。

// 对于 FT2232、FT232R、FT245R、FT2232H，其他元素需要非零值 或者

// FT4232H 设备。

FT\_PROGRAM\_DATA ftData = {

0x00000000, // 标头 - 必须是 0x00000000

0xFFFFFFFF, // 标头 - 必须是 0xffffffff 0xffffffff

0x00000004, // 标头 - FT\_程序\_数据 版本 0x0403, // 视频识别码

0x6001， // PID

（字符 \*) “FTDI”， // 制造商

（字符 \*) “金融时报”， // 制造商 ID（字符\*）“USB HS 串行 转换器”， // 描述（字符 \*) “FT000001”， // 串行 44号， // 最大功率

1, // 即插即用

0, // 自供电

1, // 远程唤醒

1, // 非零如果Rev4芯片为零 否则

0, // 如果端点为非零 等时的

0, // 如果输出端点为非零 等时的

0, // 如果下拉则非零 已启用

1, // 如果序列号为非零 用过的

0, // 如果芯片使用则非零 USB版本

0x0110, // BCD (0x0200 => USB2)

//

// FT2232C 扩展（如果版本 = 1 或更高版本则启用）

//

0, // 非零，如果 Rev5 芯片为零 否则

0, // 如果端点为非零 等时的

0, // 如果端点为非零 等时的

0, // 如果输出端点为非零 等时的

0, // 如果输出端点为非零 等时的

0, // 如果下拉则非零 已启用

0, // 如果序列号为非零 用过的

0, // 如果芯片使用则非零 USB版本

0x0, // BCD (0x0200 => USB2)

0, // 如果接口为高电平则非零 当前的

0, // 如果接口为高电平则非零 当前的

0, // 如果接口为 245，则非零 先进先出

0, // 如果接口是 245 FIFO CPU，则非零 目标

0, // 如果接口速度快则非零 连续剧

0, // 如果接口使用 VCP，则非零 司机

0, // 如果接口为 245，则非零 先进先出

0, // 如果接口是 245 FIFO CPU，则非零 目标

0, // 如果接口速度快则非零 连续剧

0, // 如果接口使用 VCP，则非零 司机

//

// FT232R 扩展（如果版本 = 2 或更高版本则启用）

//

0, // 使用外部 振荡器

0, // 高驱动 输入/输出

0, // 端点 尺寸

0, // 如果下拉则非零 已启用

0, // 如果序列号为非零 用过的

0, // 如果反转则非零 TXD

0, // 如果反转则非零 接收数据

0, // 如果反转则非零 实时传输系统

0, // 如果反转则非零 CTS

0, // 如果反转则非零 数字TR

0, // 如果反转则非零 数字SR

0, // 如果反转则非零 二氯二苯醚

0, // 如果反转则非零 RI

0, // Cbus 复用器 控制

0, // Cbus 复用器 控制

0, // Cbus 复用器 控制

0, // Cbus 复用器 控制

0, // Cbus 复用器 控制

0, // 如果使用 D2XX，则非零 司机

//

// Rev 7 (FT2232H) 扩展（如果版本 = 3 或更高版本则启用）

//

0, // 如果下拉则非零 已启用

0, // 如果序列号为非零 用过的

0, // 如果 AL 引脚速度较慢则非零 屠杀

0, // 如果 AL 引脚是施密特引脚，则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 AH 引脚速度较慢则非零 屠杀

0, // 如果 AH 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 BL 引脚速度较慢则非零 屠杀

0, // 如果 BL 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 BH 引脚速度较慢则非零 屠杀

0, // 如果 BH 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果接口为 245，则非零 先进先出

0, // 如果接口是 245 FIFO CPU，则非零 目标

0, // 如果接口速度快则非零 连续剧

0, // 如果接口使用 VCP，则非零 司机

0, // 如果接口为 245，则非零 先进先出

0, // 如果接口是 245 FIFO CPU，则非零 目标

0, // 如果接口速度快则非零 连续剧

0, // 如果接口使用 VCP，则非零 司机

0, // 如果使用 BCBUS7 来节省电量，则非零 自己-

// 动力设计

//

// Rev 8 (FT4232H) 扩展（如果版本 = 4 则启用）

//

0, // 如果下拉则非零 已启用

0, // 如果序列号为非零 用过的

0, // 如果 AL 引脚速度较慢则非零 屠杀

0, // 如果 AL 引脚是施密特引脚，则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 AH 引脚速度较慢则非零 屠杀

0, // 如果 AH 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 BL 引脚速度较慢则非零 屠杀

0, // 如果 BL 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果 BH 引脚速度较慢则非零 屠杀

0, // 如果 BH 引脚为施密特则非零 输入

0, // 有效值为 4mA、8mA、12mA, 16毫安

0, // 如果端口 A 使用 RI 作为 RS485，则非零 TXDEN

0, // 如果端口 B 使用 RI 作为 RS485，则非零 TXDEN

0, // 如果端口 C 使用 RI 作为 RS485，则非零 TXDEN

0, // 如果端口 D 使用 RI 作为 RS485，则非零 TXDEN

0, // 如果接口使用 VCP，则非零 司机

0, // 如果接口使用 VCP，则非零 司机

0, // 如果接口使用 VCP，则非零 司机

0, // 如果接口使用 VCP，则非零 司机

//

// Rev 9 (FT232H) 扩展（如果版本 = 5 则启用）

//

0, // 如果下拉则非零 已启用

0, // 如果序列号为非零 用过的

0, // 如果交流引脚速度较慢则非零 屠杀

0, // 如果 AC 引脚为施密特则非零 输入

|  |  |  |
| --- | --- | --- |
| 0, | // | 有效值为 4mA、8mA、12mA、16mA |
| 0, | // | 如果 AD 引脚转换缓慢，则非零 |
| 0, | // | 如果 AD 引脚是施密特输入，则非零 |
| 0, | // | 有效值为 4mA、8mA、12mA、16mA |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | Cbus复用器控制 |
| 0, | // | 如果接口是 245 FIFO，则非零 |
| 0, | // | 如果接口是 245 FIFO CPU 目标，则非零 |
| 0, | // | 如果接口是快速串行，则非零 |
| 0, | // | 如果接口是 FT1248，则非零 |
| 0, | // | FT1248 时钟极性 - 时钟空闲高 (1) 或 |
|  | // | 时钟空闲低电平 (0) |
| 0, | // | FT1248数据为LSB(1)或MSB(0) |
| 0, | // | FT1248流量控制使能 |
| 0, | // | 如果接口使用 VCP 驱动程序，则非零 |
| 0 | // | 如果使用 ACBUS7 来节省电量，则非零 |
|  | // | 自供电设计 |

};

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);如果（ftStatus == FT\_OK）{

ftStatus = FT\_EE\_Program(ftHandle, &ftData);如果（ftStatus == FT\_OK）{

// FT\_EE\_程序 好！

}

别的 {

}

// FT\_EE\_Program 失败！

FT\_Close(ftHandle);

}

## FT\_EE\_ProgramEx

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

分别对 EEPROM 进行编程并传递字符串。

### 定义

FT\_STATUS **FT\_EE\_ProgramEx** (FT\_HANDLE *ftHandle* , PFT\_PROGRAM\_DATA *pData* ,

char \* 制造商、 char \* 制造商 ID、 char \* 描述、 char \* 序列号）

### 参数

*ftHandle* 的句柄 设备。

*pData* 指向类型结构的指针 FT\_PROGRAM\_DATA。

*\*Manufacturer* 指向包含空终止字符串的指针 制造商

姓名。

*\*ManufacturerId* 指向包含制造商的空终止字符串的指针 ID。

*\*描述* 指向包含空终止字符串的指针 这 设备描述。

*\*SerialNumber* 指向包含设备的以 null 结尾的字符串的指针 连续剧

数字。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

[FT\_EE\_Program函数](#_bookmark55)的这种变体是为了提供对 LabVIEW 等语言的支持，当字符串指针包含在结构中时，可能会出现问题。

该函数将参数*pData解释为指向FT\_PROGRAM\_DATA*类型结构的指针，该结构包含要写入 EEPROM 的数据。数据被写入EEPROM，然后读回并验证。

*FT\_PROGRAM\_DATA结构*中的字符串指针参数应分配为 DWORD，以避免参数重叠。然后将字符串参数单独传入。

如果*SerialNumber*字段为NULL，或者*SerialNumber*指向NULL字符串，则会根据*ManufacturerId*和当前日期和时间生成序列号。制造*商*字符串长度加上*描述*字符串长度必须小于或等于 40 个字符。

请注意，必须告知 DLL正在使用哪个版本的*FT\_PROGRAM\_DATA结构。*这是通过结构的*Signature1* 、 *Signature2*和*Version元素完成的。 Signature1*应始终为*0x00000000* ， *Signature2*应始终为*0xFFFFFFFF* ，并且*Version*可以设置为使用所需的版本。为了与所有当前设备兼容，*版本*应设置为FTD2XX.h 中定义的*FT\_PROGRAM\_DATA结构的最新版本。*

如果*pData*为 NULL，则结构版本将默认为 0（原始 BM 系列），并且器件将使用默认数据进行编程：

## FT\_EE\_UAS尺寸

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取 EEPROM 用户区的可用大小。

### 定义

FT\_STATUS **FT\_EE\_UASize** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwSize* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwSize* 指向接收可用大小的 DWORD 的指针，格式为 字节， EEPROM 用户的 区域。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

FTDI 设备 EEPROM 的用户区域是 EEPROM 中未用于设备配置信息和描述符的总区域。用户可以使用该区域来存储特定于其应用程序的信息。用户区域的大小取决于编程到 EEPROM 中的*Manufacturer* 、 *ManufacturerId* 、 *Description*和*SerialNumber字符串的长度。*

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

如果（ftStatus！= FT\_OK）{

// FT\_打开失败！

}

DWORD EEUA\_Size；

ftStatus = FT\_EE\_UASize(ftHandle, &EEUA\_Size);如果（ftStatus == FT\_OK）{

// FT\_EE\_UASize 确定

// EEUA\_Size 包含 EEUA 的大小（以字节为单位）

}

别的 {

}

// FT\_EE\_UASize 失败！

FT\_Close(ftHandle);

## FT\_EE\_UA读

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

读取EEPROM用户区的内容。

### 定义

FT\_STATUS **FT\_EE\_UARead** （FT\_HANDLE *ftHandle* ，PUCHAR *pucData* ，DWORD *dwDataLen* ，LPDWORD lpdwBytesRead）

### 参数

*ftHandle* 的句柄 设备。

*pucData* 指向包含要存储的数据的缓冲区的指针 读。

*dwDataLen* 包含数据存储的缓冲区的大小（以字节为单位） 到 被阅读。

*lpdwBytesRead* 指向接收字节数的 DWORD 的指针 读。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*pucData解释为指向大小为dwDataLen 的*字节数组的指针，该数组包含要从 EEPROM 用户区域读取的数据的存储空间。实际读取的字节数存储在*lpdwBytesRead引用的 DWORD 中*。

如果*dwDataLen*小于 EEPROM 用户区域的大小，则将*dwDataLen*字节读入缓冲区。否则，整个 EEPROM 用户区将被读入缓冲区。可用的用户区域大小可以通过调用[FT\_EE\_UASize](#_bookmark57)来确定。

当 FT\_EE\_UARead 返回时，应用程序应检查函数返回值和*lpdwBytesRead 。*

### 例子

FT\_HANDLE ftHandle;

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

如果（ftStatus！= FT\_OK）{

// FT\_打开失败！

}

无符号字符缓冲区[64]； DWORD 字节读取；

ftStatus = FT\_EE\_UARead(ftHandle, 缓冲区, 64, &BytesRead);如果（ftStatus == FT\_OK）{

// FT\_EE\_UA读取成功

// 用户区数据存放在Buffer中

// 从 EEUA 读取的字节数存储在 BytesRead 中

}

别的 {

}

// FT\_EE\_UA读取失败！

FT\_Close(ftHandle);

## FT\_EE\_UA写入

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将数据写入EEPROM用户区。

### 定义

FT\_STATUS **FT\_EE\_UAWrite** (FT\_HANDLE *ftHandle* 、 PUCHAR *pucData* 、 DWORD *dwDataLen* )

### 参数

*ftHandle* 的句柄 设备。

*pucData* 指向包含要写入的数据的缓冲区的指针 书面。

*dwDataLen* 包含数据存储的缓冲区的大小（以字节为单位） 到 被阅读。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*pucData解释为指向大小为dwDataLen 的*字节数组的指针，该数组包含要写入 EEPROM 用户区域的数据。 *dwDataLen*大于 EEPROM 用户区的大小是编程错误。可用的用户区域大小可以通过调用[FT\_EE\_UASize](#_bookmark57)来确定。

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

如果（ftStatus！= FT\_OK）{

// FT\_打开失败！

}

char \*buffer = (char \*) "你好，世界";

ftStatus = FT\_EE\_UAWrite(ftHandle, (unsigned char\*)buffer, 12); if(ftStatus!= FT\_OK) {

// FT\_EE\_UAWRITE 失败

}

别的 {

}

// FT\_EE\_UAWRITE 失败

FT\_Close(ftHandle);

## FT\_EEPROM\_读取

### 支持的操作系统

Windows（XP 及更高版本） Linux

### 概括

从EEPROM读取数据，该命令适用于所有现有的FTDI芯片组，并且必须用于FT-X系列。

### 定义

FT\_STATUS **FT\_EEPROM\_Read** (FT\_HANDLE ftHandle, void \*eepromData, DWORD eepromDataSize,

char \* 制造商、 char \* 制造商 ID、 char \* 描述、 char \* 序列号）；

### 参数

*ftHandle* 的句柄 设备。

*\*eepromData* 指向包含数据的缓冲区的指针 是 读。注意：这个结构对于每个设备都是不同的 类型。

*epromDataSize* 包含要存储的数据的 eepromData 缓冲区的大小 是

读。

*\*Manufacturer* 指向包含空终止字符串的指针 制造商

姓名

*\*ManufacturerId* 指向包含制造商的空终止字符串的指针 ID。

*\*描述* 指向包含空终止字符串的指针 这 设备描述。

*\*SerialNumber* 指向包含设备的以 null 结尾的字符串的指针 连续剧

数字。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*\*eepromDATA解释*为指向与正在访问的设备类型匹配的结构的指针，例如

*PFT\_EEPROM\_232B*是 FT2xxB 器件的结构。 *PFT\_EEPROM\_2232*是 FT2232D 器件的结构。 *PFT\_EEPROM\_232R*是 FT232R 器件的结构。 *PFT\_EEPROM\_2232H*是 FT2232H 器件的结构。 *PFT\_EEPROM\_4232H*是 FT4232H 器件的结构。 *PFT\_EEPROM\_232H*是 FT232H 器件的结构。

*PFT\_EEPROM\_X\_SERIES*是 FT2xxX 器件的结构。

该函数不对缓冲区大小执行任何检查，因此缓冲区在 eepromDATA 中*传递*

结构必须足够大以容纳它们各自的字符串（包括空终止符）。

以下示例中显示的尺寸绰绰有余，如有必要可以向下舍入。限制是*制造商*字符串长度加上*描述*字符串长度小于或等于 40 个字符。

请注意，必须告知 DLL正在使用哪个版本的*eepromDATA结构。*这是通过*PFT\_EEPROM\_HEADER*结构完成的。该结构的第一个元素是 deviceType，可以是 FT\_DEVICE\_BM、FT\_DEVICE\_AM、FT\_DEVICE\_2232C、FT\_DEVICE\_232R、FT\_DEVICE\_2232H、FT\_DEVICE\_4232H、FT\_DEVICE\_232H 或 FT\_DEVICE\_X\_SERIES（如 FTD2XX.h 中定义）。

**示例**FT\_HANDLE fhandle; FT\_STATUS 状态；

字符制造商[64]；字符制造商Id[64]；字符描述[64]；字符序列号[64]；

FT\_EEPROM\_HEADER ft\_eeprom\_header；

ft\_eeprom\_header.deviceType = FT\_DEVICE\_2232H； // FTxxxx 设备类型 是 访问 FT\_EEPROM\_2232H ft\_eeprom\_2232h；

ft\_eeprom\_2232h.common = ft\_eeprom\_header; ft\_eeprom\_2232h.common.deviceType = FT\_DEVICE\_2232H;

状态 = FT\_Open(0, &fthandle); if(状态！= FT\_OK)

printf("打开状态不正常 %d\n", status);

状态 = FT\_EEPROM\_Read(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), 制造商,制造商 ID, 描述, 序列号);

if（状态！= FT\_OK）

printf("EEPROM\_Read 状态不正常 %d\n", status);

别的

{

}

printf("供应商ID = 0x%04x\n", ft\_eeprom\_2232h.common.VendorId); printf("产品ID = 0x%04x\n", ft\_eeprom\_2232h.common.ProductId);

……

……

FT\_Close(fthandle);

## FT\_EEPROM\_程序

### 支持的操作系统

Windows（XP 及更高版本） Linux

### 概括

将数据写入EEPROM，该命令适用于所有现有的FTDI芯片组，并且必须用于FT-X系列。

### 定义

FT\_STATUS **FT\_EEPROM\_Program** (FT\_HANDLE ftHandle, void \*eepromData, DWORD eepromDataSize,

char \* 制造商、 char \* 制造商 ID、 char \* 描述、 char \* 序列号）；

### 参数

*ftHandle* 的句柄 设备。

*\*eepromData* 指向包含数据的缓冲区的指针 是 书面。注意：这个结构对于每个设备都是不同的 类型。

*epromDataSize* 包含要存储的数据的 eepromData 缓冲区的大小 是

书面。

*\*Manufacturer* 指向包含空终止字符串的指针 制造商

姓名

*\*ManufacturerId* 指向包含制造商的空终止字符串的指针 ID。

*\*描述* 指向包含空终止字符串的指针 这 设备描述。

*\*SerialNumber* 指向包含设备的以 null 结尾的字符串的指针 连续剧

数字。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

该函数将参数*\*eepromDATA解释*为指向与正在访问的设备类型匹配的结构的指针，例如

*PFT\_EEPROM\_232B*是 FT2xxB 器件的结构。 *PFT\_EEPROM\_2232*是 FT2232D 器件的结构。 *PFT\_EEPROM\_232R*是 FT232R 器件的结构。 *PFT\_EEPROM\_2232H*是 FT2232H 器件的结构。 *PFT\_EEPROM\_4232H*是 FT4232H 器件的结构。 *PFT\_EEPROM\_232H*是 FT232H 器件的结构。

*PFT\_EEPROM\_X\_SERIES*是 FT2xxX 器件的结构。

该函数不对缓冲区大小执行任何检查，因此缓冲区在 eepromDATA 中*传递*

结构必须足够大以容纳它们各自的字符串（包括空终止符）。

以下示例中显示的尺寸绰绰有余，如有必要可以向下舍入。限制是*制造商*字符串长度加上*描述*字符串长度小于或等于 40 个字符。

请注意，必须告知 DLL正在使用哪个版本的*eepromDATA结构。*这是通过*PFT\_EEPROM\_HEADER*结构完成的。该结构的第一个元素是 deviceType，可以是 FT\_DEVICE\_BM、FT\_DEVICE\_AM、FT\_DEVICE\_2232C、FT\_DEVICE\_232R、FT\_DEVICE\_2232H、FT\_DEVICE\_4232H、FT\_DEVICE\_232H 或 FT\_DEVICE\_X\_SERIES（如 FTD2XX.h 中定义）。

**示例**FT\_HANDLE fhandle; FT\_STATUS 状态；

字符制造商[64]；字符制造商Id[64]；字符描述[64]；字符序列号[64]；

FT\_EEPROM\_HEADER ft\_eeprom\_header；

ft\_eeprom\_header.deviceType = FT\_DEVICE\_2232H； // FTxxxx 设备类型 是 访问 FT\_EEPROM\_2232H ft\_eeprom\_2232h；

ft\_eeprom\_2232h.common = ft\_eeprom\_header; ft\_eeprom\_2232h.common.deviceType = FT\_DEVICE\_2232H;

状态 = FT\_Open(0, &fthandle);

if(状态！= FT\_OK)

printf("打开状态不正常 %d\n", status);

状态 = FT\_EEPROM\_Read(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), 制造商, 制造商 ID, 描述, 序列号);

strcpy\_s(序列号, (const char \*) "FT000001");

状态 = FT\_EEPROM\_Program(fthandle,&ft\_eeprom\_2232h, sizeof(ft\_eeprom\_2232h), 制造商, 制造商 ID, 描述, 序列号);

FT\_Close(fthandle);

# 扩展API 功能

扩展 API 函数不适用于 FT8U232AM 或 FT8U245AM 设备。 FTDI 的其他 USB-UART 和 USB-FIFO IC（FT2232H、FT4232H、FT232R、FT245R、FT2232、FT232B 和 FT245B）支持

这些功能。请注意，其中一些函数存在设备依赖性。

## FT\_设置延迟定时器

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

设置延迟计时器值。

### 定义

FT\_STATUS **FT\_SetLatencyTimer** (FT\_HANDLE *ftHandle* , UCHAR *ucTimer* )

### 参数

*ftHandle* 的句柄 设备。

*ucTimer* 延迟计时器所需的值（以毫秒为单位）。有效的 范围 是 2 – 255.

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

在 FT8U232AM 和 FT8U245AM 器件中，用于刷新接收缓冲区中剩余数据的接收缓冲区超时固定为 16 ms。在所有其他 FTDI 设备中，此超时是可编程的，并且可以在 2ms 到 255 ms 之间以 1 ms 间隔设置。这使得设备能够针对需要短数据包更快响应时间的协议进行更好的优化。

### 例子

FT\_HANDLE ftHandle； FT\_STATUS ft状态； UCHAR 延迟计时器 = 10；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetLatencyTimer(ftHandle, LatencyTimer);如果（ftStatus == FT\_OK）{

// LatencyTimer 设置为 10 毫秒

}

别的 {

}

// FT\_SetLatencyTimer 失败！

FT\_Close(ftHandle);

## FT\_GetLatencyTimer

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取延迟计时器的当前值。

### 定义

FT\_STATUS **FT\_GetLatencyTimer** (FT\_HANDLE *ftHandle* , PUCHAR *pucTimer* )

### 参数

*ftHandle* 的句柄 设备。

*pucTimer* 指向 unsigned char 的指针，用于存储延迟计时器 价值。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

在 FT8U232AM 和 FT8U245AM 器件中，用于刷新接收缓冲区中剩余数据的接收缓冲区超时固定为 16 ms。在所有其他 FTDI 设备中，此超时是可编程的，并且可以在 2ms 到 255 ms 之间以 1 ms 间隔设置。这使得设备能够针对需要短数据包更快响应时间的协议进行更好的优化。

### 例子

FT\_HANDLE ftHandle； FT\_STATUS ft状态； UCHAR 延迟计时器；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_GetLatencyTimer(ftHandle, &LatencyTimer);如果（ftStatus == FT\_OK）{

// LatencyTimer 包含当前值

}

别的 {

}

// FT\_GetLatencyTimer 失败！

FT\_Close(ftHandle);

## FT\_设置位模式

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

启用不同的芯片模式。

### 定义

FT\_STATUS **FT\_SetBitMode** (FT\_HANDLE *ftHandle* 、 UCHAR *ucMask* 、 UCHAR *ucMode* )

### 参数

*ftHandle* 的句柄 设备。

*ucMask*位模式掩码所需的值。这设置了哪些位是输入和输出。位值 0 设置相应的 别针 对于 输入，位值 1 将相应的引脚设置为 输出。

在 CBUS Bit Bang 的情况下，该值的高半字节控制哪些引脚是输入和输出，而低半字节控制哪些输出是高电平和低电平。

*ucMode* 模式值。可以是其中之一 这 以下：0x0 = 重置

0x1 = 异步位爆炸

0x2 = MPSSE（仅限 FT2232、FT2232H、FT4232H 和 FT232H 器件）

0x4 = 同步 Bit Bang（仅限 FT232R、FT245R、FT2232、FT2232H、FT4232H 和 FT232H 器件） 0x8 = MCU 主机总线仿真模式（仅限 FT2232、FT2232H、FT4232H 和 FT232H 器件）

0x10 = 快速光隔离串行模式（仅限 FT2232、FT2232H、FT4232H 和 FT232H 器件） 0x20 = CBUS Bit Bang 模式（仅限 FT232R 和 FT232H 器件）

0x40 = 单通道同步 245 FIFO 模式（仅限 FT2232H 和 FT232H 器件）

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

有关 FT232R 可用位模式的说明，请参阅应用笔记“FT232R 和 FT245R 的位 Bang 模式”。

有关 FT2232 可用位模式的说明，请参阅应用笔记“FT2232 的位模式功能”。

有关 FT232B 和 FT245B Bit Bang 模式的说明，请参阅应用笔记“FT232B/FT245B Bit Bang 模式”。

应用说明可从 FTDI 网站下载。

请注意，要将 CBUS Bit Bang 用于 FT232R，必须在 EEPROM 中将 CBUS 配置为 CBUS Bit Bang。

请注意，要对 FT2232H 使用单通道同步 245 FIFO 模式，通道 A 必须在 EEPROM 中配置为 FT245 FIFO 模式。

### 例子

FT\_HANDLE ftHandle； FT\_STATUS ft状态； UCHAR 掩码 = 0xff；

UCHAR 模式 = 1； // 设置异步bit-bang模式

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetBitMode(ftHandle, 掩码, 模式);如果（ftStatus == FT\_OK）{

// 0xff 写入设备

}

别的 {

}

// FT\_SetBitMode 失败！

FT\_Close(ftHandle);

## FT\_获取位模式

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取数据总线的瞬时值。

### 定义

FT\_STATUS **FT\_GetBitMode** (FT\_HANDLE *ftHandle* , PUCHAR *pucMode* )

### 参数

*ftHandle* 的句柄 设备。

*pucMode* 指向用于存储瞬时值的无符号字符的指针 数据 总线值。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

有关 FT232R 可用位模式的说明，请参阅应用笔记“FT232R 和 FT245R 的位 Bang 模式”。

有关 FT2232 可用位模式的说明，请参阅应用笔记“FT2232 的位模式功能”。

有关 FT232B 和 FT245B Bit Bang 模式的说明，请参阅应用笔记“FT232B/FT245B Bit Bang Mode”。

有关 FT4232H 和 FT2232H 器件支持的位模式的说明，请参阅 IC 数据表。

这些应用说明可从 FTDI 网站下载。

### 例子

FT\_HANDLE ftHandle；

UCHAR 位模式；

FT\_STATUS ft状态；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_GetBitMode(ftHandle, &BitMode);如果（ftStatus == FT\_OK）{

// BitMode 包含当前值

}

别的 {

}

// FT\_GetBitMode 失败！

FT\_Close(ftHandle);

## FT\_设置USB参数

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

设置 USB 请求传输大小。

### 定义

FT\_STATUS **FT\_SetUSBParameters** （FT\_HANDLE *ftHandle* 、DWORD *dwInTransferSize* 、DWORD dwOutTransferSize）

### 参数

*ftHandle* 的句柄 设备。

*dwInTransferSize* USB IN 的传输大小 要求。

*dwOutTransferSize* USB OUT 的传输大小 要求。

### 返回值

如果成功则返回 FT\_OK，否则返回值为 FT 错误代码。

### 评论

此函数可用于更改默认传输大小 4096 字节的传输大小，以更好地满足应用程序要求。传输大小必须设置为 64 字节和 64k 字节之间的 64 字节的倍数。

当调用 FT\_SetUSBParameters 时，更改立即生效，并且更改时驱动程序中保存的所有数据都会丢失。

请注意，目前仅支持*dwInTransferSize 。*

### 例子

FT\_HANDLE ftHandle；

FT\_STATUS ft状态；

双字输入传输大小 = 16384；

ftStatus = FT\_Open(0, &ftHandle); if(ftStatus!= FT\_OK) {

// FT\_Open 失败返回;

}

ftStatus = FT\_SetUSBParameters(ftHandle, InTransferSize, 0);如果（ftStatus == FT\_OK）{

// 传输大小设置为 16 KB

}

别的 {

}

// FT\_SetUSBParameters 失败！

FT\_Close(ftHandle);

# FT-Win32 API 功能

本节中的函数用于轻松从 Win32 串行端口应用程序进行移植。这些功能在非 Windows 平台下受支持，以协助从 Windows 移植现有应用程序。请注意，除非另有说明，否则经典 D2XX 函数和 Win32 D2XX 函数不应混合使用。

## FT\_W32\_创建文件

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

打开指定的设备并返回一个句柄，该句柄将用于后续访问。装置

可以通过其序列号、设备描述或位置来指定。如果需要重叠 I/O，则必须使用此功能。

### 定义

FT\_HANDLE **FT\_W32\_CreateFile** （PVOID *pvArg1* ，DWORD *dwAccess* ，DWORD *dwShareMode* ，

LPSECURITY\_ATTRIBUTES *lpSecurityAttributes* 、DWORD *dwCreate* 、DWORD dwAttrsAndFlags、HANDLE hTemplate）

### 参数

*pvArg1*的含义取决于*dwAttrsAndFlags的值*。可以是指向包含设备描述或序列号的以 null 结尾的字符串的指针，也可以是设备的位置。这些值可以从[FT\_CreateDeviceInfoList](#_bookmark6) 、 [FT\_GetDeviceInfoDetail](#_bookmark8)或[FT\_ListDevices](#_bookmark9)获取 功能。

*dwAccess* 设备访问类型。访问可以 是 GENERIC\_READ、GENERIC\_WRITE 或两者。被忽略于 Linux。

*dwShareMode* 设备如何共享。该值必须设置为 0。

*lpSecurityAttributes* 该参数无效，应设置为 无效的。

*dwCreate* 该参数必须设置为 OPEN\_EXISTING。被忽略于 Linux。

*dwAttrsAndFlags* 文件属性和标志。这个参数是一个 组合 FILE\_ATTRIBUTE\_NORMAL、FILE\_FLAG\_OVERLAPPED（如果使用重叠 I/O）、 [FT\_OPEN\_BY\_SERIAL\_NUMBER](#_bookmark93) （如果*lpszName*是设备的序列号）和[FT\_OPEN\_BY\_DESCRIPTION （](#_bookmark93)如果*lpszName*是设备的序列号） 描述。

*hTemplate* 该参数必须是 无效的。

### 返回值

如果函数成功，返回值是一个句柄。

如果函数不成功，则返回值为 Win32 错误代码 INVALID\_HANDLE\_VALUE。

### 评论

pvArg1的含义取决于dwAttrsAndFlags：如果在dwAttrsAndFlags中设置了[*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93)或[*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93) ，则pvArg1包含一个指向以空结尾的字符串的指针，该字符串包含设备的序列号或描述；如果dwAttrsAndFlags 中设置了[*FT\_OPEN\_BY\_LOCATION*](#_bookmark93) ，则 pvArg1 被解释为包含设备位置 ID 的 long 类型值。

dwAccess 可以是 GENERIC\_READ、GENERIC\_WRITE 或两者； dwShareMode 必须设置为 0； lpSecurityAttributes 必须设置为 NULL； dwCreate 必须设置为 OPEN\_EXISTING； dwAttrsAndFlags 是 FILE\_ATTRIBUTE\_NORMAL、FILE\_FLAG\_OVERLAPPED（如果使用重叠 I/O）、 [*FT\_OPEN\_BY\_SERIAL\_NUMBER*](#_bookmark93)或[*FT\_OPEN\_BY\_DESCRIPTION*](#_bookmark93)或[*FT\_OPEN\_BY\_LOCATION*](#_bookmark93)的组合； h模板

必须为 NULL。

请注意，Linux、Mac OS X 和 Windows CE 不支持重叠 IO 或位置 ID。

### 例子

下面的示例使用这些变量。

FT\_STATUS ft状态； FT\_HANDLE ftHandle；字符缓冲区[64]；

1. 使用串行端口打开重叠 I/O 设备 数字

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_SERIAL\_NUMBER);

ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_现有，

文件属性正常 |文件标志重叠 | FT\_OPEN\_BY\_SERIAL\_NUMBER，

0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice 失败

1. 使用其打开非重叠 I/O 设备 描述

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_DESCRIPTION);

ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_现有，

文件属性正常 | FT\_OPEN\_BY\_DESCRIPTION, 0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice 失败

1. 使用其打开非重叠 I/O 设备 地点

长locID；

ftStatus = FT\_ListDevices(0,&locID,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_LOCATION); ftHandle = FT\_W32\_CreateFile((LPCTSTR) locID,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_现有，

文件属性正常 | FT\_OPEN\_BY\_LOCATION, 0);

if (ftHandle == INVALID\_HANDLE\_VALUE)

; // FT\_W32\_CreateDevice 失败

## FT\_W32\_CloseHandle

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

关闭指定的设备句柄。

### 定义

BOOL **FT\_W32\_CloseHandle** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

此示例展示了如何使用其描述在打开设备以进行非重叠 I/O 后关闭该设备。

FT\_STATUS ft状态； FT\_HANDLE ftHandle；字符缓冲区[64]；

ftStatus = FT\_ListDevices(0,Buf,FT\_LIST\_BY\_INDEX|FT\_OPEN\_BY\_DESCRIPTION); ftHandle = FT\_W32\_CreateFile((LPCTSTR) Buf,GENERIC\_READ|GENERIC\_WRITE,0,0,

OPEN\_现有，

文件属性正常 | FT\_OPEN\_BY\_DESCRIPTION, 0);

如果（ftHandle == INVALID\_HANDLE\_VALUE）{

// FT\_W32\_CreateDevice 失败

}

别的 {

}

// FT\_W32\_CreateFile 好的，所以做一些工作，最终... FT\_W32\_CloseHandle(ftHandle);

## FT\_W32\_读文件

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）

### 概括

从设备读取数据。

### 定义

BOOL **FT\_W32\_ReadFile** (FT\_HANDLE *ftHandle* , LPVOID *lpBuffer* , DWORD *dwBytesToRead* ,

LPDWORD lpdwBytes返回，LPOVERLAPPED lpOverlapped)

### 参数

*ftHandle* 的句柄 设备。

*lpBuffer* 指向从缓冲区接收数据的指针 设备。

*dwBytesToRead* 要读取的字节数 设备。

*lpdwBytesReturned* 指向接收字节数的变量的指针 读 来自 设备。

*lpOverlapped* 指向重叠的指针 结构。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

该函数支持非重叠和重叠 I/O，Linux、Mac OS X 和

仅支持非重叠 IO 的 Windows CE。

非重叠 I/O

对于非重叠 I/O ，参数*lpOverlapped必须为 NULL。*

*lpdwBytesReturned*中读取的字节数。

*dwBytesToRead*被读入缓冲区之前，该函数不会返回。接收队列中的字节数可以通过调用[FT\_GetStatus来确定](#_bookmark31) 或[FT\_GetQueueStatus](#_bookmark26) ，并作为*dwBytesToRead传递*，以便该函数读取设备并立即返回。

如果在先前调用[FT\_W32\_SetCommTimeouts中设置了读取超时](#_bookmark79)，则该函数将在计时器到期或已读取*dwBytesToRead时返回（以先发生者为准）。*如果发生超时，任何可用数据都会被读入*lpBuffer*并且该函数返回一个非零值。

在处理缓冲区时应使用函数返回值和*lpdwBytesReturned 。*如果返回值非零并且*lpdwBytesReturned*等于*dwBytesToRead*则函数已正常完成。如果返回值非零且*lpdwBytesReturned*小于*dwBytesToRead* ，则发生超时，并且读取请求已部分完成。请注意，如果发生超时并且没有读取数据，则返回值仍为非零。

返回值*FT\_IO\_ERROR*表明函数参数存在错误，或者发生了 USB 断开连接等致命错误。

重叠 I/O

当设备已打开以进行重叠 I/O 时，应用程序可以发出请求并在请求挂起时执行一些附加工作。这与非重叠 I/O 的情况形成对比，在非重叠 I/O 中，应用程序发出请求并仅在请求完成后才再次接收控制。

参数*lpOverlapped*必须指向已初始化的 OVERLAPPED 结构。

如果接收队列中有足够的数据来满足请求，则请求立即完成并且返回代码不为零。读取的字节数在*lpdwBytesReturned中返回*。

如果接收队列中没有足够的数据来满足请求，则请求立即完成，并且返回码为零，表示发生错误。应用程序应调用[FT\_W32\_GetLastError](#_bookmark87)来获取错误原因。如果错误代码为ERROR\_IO\_PENDING，则重叠操作仍在进行中，应用程序可以执行其他处理。最终，应用程序通过调用[FT\_W32\_GetOverlappedResult](#_bookmark73)检查重叠请求的结果。

如果成功，读取的字节数将在*lpdwBytesReturned中返回*。

### 例子

1. 此示例演示如何使用非重叠从设备读取 256 字节 输入/输出。

FT\_HANDLE ftHandle； // 通过 FT\_W32\_CreateFile 设置非重叠 I/O FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

字符缓冲区[256]；

双字 dwToRead = 256;双字 dwRead；

DWORD osRead；

if (FT\_W32\_ReadFile(ftHandle, Buf, dwToRead, &dwRead, (LPOVERLAPPED) &osRead)) { if (dwToRead == dwRead){

// FT\_W32\_ReadFile 确定

}

别的{

}

别的{

}

// FT\_W32\_ReadFile 超时

// FT\_W32\_ReadFile 失败

}

1. 此示例演示如何使用重叠 I/O 从设备读取 256 字节。 FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);字符缓冲区[256]；

双字 dwToRead = 256;双字 dwRead；

重叠的 osRead = { 0 };

osRead.hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

if (!FT\_W32\_ReadFile(ftHandle, Buf, dwToRead, &dwRead, &osRead)) { if (FT\_W32\_GetLastError(ftHandle) == ERROR\_IO\_PENDING) {

// 写入被延迟，所以做一些其他的事情直到...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osRead, &dwRead, 错误的））{

// 错误

}

别的 {

如果（dwToRead == dwRead）{

// FT\_W32\_ReadFile 确定

}

}

}

别的 {

}

别的{

}

// FT\_W32\_ReadFile 超时

// FT\_W32\_ReadFile 确定

}

关闭句柄（osRead.hEvent）；

## FT\_W32\_WriteFile

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将数据写入设备。

### 定义

BOOL **FT\_W32\_WriteFile** (FT\_HANDLE *ftHandle* , LPVOID *lpBuffer* , DWORD *dwBytesToWrite* ,

LPDWORD lpdwBytesWritten、LPOVERLAPPED lpOverlapped)

### 参数

*ftHandle* 的句柄 设备。

*lpBuffer* 指向包含要写入的数据的缓冲区的指针 设备。

*dwBytesToWrite* 要写入的字节数 设备。

*lpdwBytesWritten* 指向接收字节数的变量的指针 书面 到 设备。

*lpOverlapped* 指向重叠的指针 结构。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

该函数支持非重叠和重叠 I/O，Linux、Mac OS X 和

仅支持非重叠 IO 的 Windows CE。非重叠 I/O

对于非重叠 I/O ，参数*lpOverlapped必须为 NULL。*

*lpdwBytesWritten*中写入的字节数。

*dwBytesToWrite*写入设备之前，该函数不会返回。

[FT\_W32\_SetCommTimeouts 的](#_bookmark79)调用中设置了写入超时，则当计时器到期或*dwBytesToWrite*已写入（以先发生者为准）时，此函数将返回。如果发生超时， *lpdwBytesWritten*包含实际写入的字节数，并且该函数返回一个

非零值。

应用程序应始终使用函数返回值和*lpdwBytesWritten* 。如果返回值非零且*lpdwBytesWritten*等于*dwBytesToWrite* ，则函数已正常完成。如果返回值非零且*lpdwBytesWritten*小于*dwBytesToWrite* ，则发生超时，并且写入请求已部分完成。请注意，如果发生超时并且没有写入数据，则返回值仍为非零。

重叠 I/O

当设备已打开以进行重叠 I/O 时，应用程序可以发出请求并在请求挂起时执行一些附加工作。这与非重叠 I/O 的情况形成对比

应用程序发出请求并仅在请求完成后才再次接收控制。

参数*lpOverlapped*必须指向已初始化的 OVERLAPPED 结构。

该函数立即完成，返回码为零，表示发生错误。应用程序应调用[FT\_W32\_GetLastError](#_bookmark87)来获取错误原因。如果错误代码为ERROR\_IO\_PENDING，则重叠操作仍在进行中，应用程序可以执行其他处理。

[FT\_W32\_GetOverlappedResult](#_bookmark73)检查重叠请求的结果。

如果成功，写入的字节数将在*lpdwBytesWritten中返回*。

### 例子

1. 此示例显示如何使用非重叠向设备写入 128 字节 输入/输出。

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

// 通过 FT\_W32\_CreateFile 设置非重叠 i/o char Buf[128]; // 包含要写入设备的数据 DWORD dwToWrite = 128;

DWORD dw写入；

DWORD osWrite；

if (FT\_W32\_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, (LPOVERLAPPED) &osWrite)) { if (dwToWrite == dwWritten){

// FT\_W32\_WriteFile 确定

}

别的{

}

别的{

}

// FT\_W32\_WriteFile 超时

// FT\_W32\_WriteFile 失败

}

1. 此示例说明如何使用重叠 I/O 将 128 字节写入设备。 FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

// 通过 FT\_W32\_CreateFile 设置重叠 I/O

字符缓冲区[128]； // 包含要写入设备的数据 DWORD dwToWrite = 128;

DWORD dw写入；重叠的 osWrite = { 0 };

if (!FT\_W32\_WriteFile(ftHandle, Buf, dwToWrite, &dwWritten, &osWrite)) { if (FT\_W32\_GetLastError(ftHandle) == ERROR\_IO\_PENDING) {

// 写入被延迟，所以做一些其他的事情直到...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osWrite, &dwWritten, FALSE)){

// 错误

}

别的 {

如果（dwToWrite == dwWritten）{

// FT\_W32\_WriteFile 确定

}

别的{

}

}

}

// FT\_W32\_WriteFile 超时

}

别的 {

}

// FT\_W32\_WriteFILE 确定

## FT\_W32\_GetOverlappedResult

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取重叠操作的结果。

### 定义

BOOL **FT\_W32\_GetOverlappedResult** (FT\_HANDLE *ftHandle* , LPOVERLAPPED *lpOverlapped* ,

LPDWORD lpdwBytesTransferred, BOOL b 等待)

### 参数

*ftHandle* 的句柄 设备。

*lpOverlapped* 指向重叠的指针 结构。

*lpdwBytesTransferred* 指向接收字节数的变量的指针 字节 重叠期间转移 手术。

*bWait* 如果函数直到指定时间才返回，则设置为 TRUE 手术 已经 完全的。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

此函数与重叠 I/O 一起使用，因此在 Linux、Mac OS X 或 Windows CE 中不支持。为了

其使用说明，请参阅[FT\_W32\_ReadFile](#_bookmark71)和[FT\_W32\_WriteFile](#_bookmark72) 。

## FT\_W32\_EscapeCommFunction

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

执行扩展功能。

### 定义

BOOL **FT\_W32\_EscapeCommFunction** (FT\_HANDLE *ftHandle* , DWORD *dwFunc* )

### 参数

*ftHandle* 的句柄 设备。

*dwFunc* 要执行的扩展函数可以是以下之一 这 以下值：

CLRDTR – 清除 DTR 信号 CLRRTS – 清除 RTS 信号 SETDTR – 设置 DTR 信号 SETRTS – 设置 RTS 信号 SETBREAK – 设置 BREAK 状况

CLRBREAK – 清除 BREAK 条件

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle； // 通过 FT\_W32\_CreateFile 设置

FT\_W32\_EscapeCommFunction(ftHandle,CLRDTR); // 清除DTR信号 FT\_W32\_EscapeCommFunction(ftHandle,SETRTS); // 设置RTS信号

## FT\_W32\_GetCommModemStatus

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数获取当前调制解调器控制值。

### 定义

BOOL **FT\_W32\_GetCommModemStatus** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwStat* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwStat* 指向包含调制解调器控制值的变量的指针。 这 调制解调器控制值可以是以下组合 下列的：

MS\_CTS\_ON – 清除发送 (CTS) 已开启 MS\_DSR\_ON – 数据集就绪 (DSR) 已开启

MS\_RING\_ON – 振铃指示器 (RI) 开启

MS\_RLSD\_ON – 接收线路信号检测 (RLSD) 开启

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle； // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwStatus；

如果 (FT\_W32\_GetCommModemStatus(ftHandle,(LPDWORD)&dwStatus)) {

// FT\_W32\_GetCommModemStatus 正常 if (dwStatus & MS\_CTS\_ON)

; // CTS 开启

if (dwStatus & MS\_DSR\_ON)

; // DSR 已开启

if (dwStatus & MS\_RING\_ON)

; // RI 已开启

if (dwStatus & MS\_RLSD\_ON)

; // RLSD 开启

}

别的

; // FT\_W32\_GetCommModemStatus 失败

## FT\_W32\_SetupComm

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置读取和写入缓冲区。

### 定义

BOOL **FT\_W32\_SetupComm** (FT\_HANDLE *ftHandle* , DWORD *dwReadBufferSize* ,

DWORD dwWriteBufferSize)

### 参数

*ftHandle* 的句柄 设备。

*dwReadBufferSize 的* 长度（以字节为单位） 读 缓冲。 *dwWriteBufferSize 的* 长度（以字节为单位） 写 缓冲。**返回 价值**

如果函数成功，则返回值非零。

如果函数不成功，则返回值为零。

### 评论

该功能没有任何作用。驱动程序有责任为 I/O 请求分配足够的存储空间。

## FT\_W32\_SetCommState

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数根据设备控制块（DCB）的内容设置设备的状态。

### 定义

BOOL **FT\_W32\_SetCommState** (FT\_HANDLE *ftHandle* , LPFTDCB *lpftDcb* )

### 参数

*ftHandle* 的句柄 设备。

*lpftDcb* 指向 FTDCB 的指针 结构。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle；

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);// 通过 FT\_W32\_CreateFile 设置 FTDCB ftDCB;

如果 (FT\_W32\_GetCommState(ftHandle,&ftDCB)) {

// FT\_W32\_GetCommState ok，设备状态位于 ftDCB ftDCB.BaudRate = 921600; //改变波特率

if (FT\_W32\_SetCommState(ftHandle,&ftDCB))

; // FT\_W32\_SetCommState 确定

}

别的

别的

; // FT\_W32\_SetCommState 失败

; // FT\_W32\_GetCommState 失败

## FT\_W32\_GetCommState

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本）

Windows CE（4.2 及更高版本）

### 概括

该函数获取当前设备状态。

### 定义

BOOL **FT\_W32\_GetCommState** (FT\_HANDLE *ftHandle* , LPFTDCB *lpftDcb* )

### 参数

*ftHandle* 的句柄 设备。

*lpftDcb* 指向 FTDCB 的指针 结构。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

设备的当前状态在设备控制块中返回。

### 例子

FT\_HANDLE ftHandle； // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FTDCB ftDCB； if (FT\_W32\_GetCommState(ftHandle,&ftDCB))

; // FT\_W32\_GetCommState ok，设备状态在 ftDCB 中

别的

; // FT\_W32\_GetCommState 失败

## FT\_W32\_SetCommTimeouts

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数设置 I/O 请求的超时参数。

### 定义

BOOL **FT\_W32\_SetCommTimeouts** (FT\_HANDLE *ftHandle* , LPFTTIMEOUTS *lpftTimeouts* )

### 参数

*ftHandle* 的句柄 设备。

*lpftTimeouts* 指向 FTTIMEOUTS 结构的指针，用于存储超时 信息。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

超时是使用 FTTIMEOUTS 结构中的信息计算的。

对于读取请求，要读取的字节数乘以总超时乘数，并添加到总超时常量中。因此，如果 TS 是 FTTIMEOUTS 结构，并且要读取的字节数为 dwToRead，则读取超时 rdTO 计算如下。

rdTO = (dwToRead \* TS.ReadTotalTimeoutMultiplier) + TS.ReadTotalTimeoutConstant

对于写请求，要写入的字节数乘以总超时乘数，并添加到总超时常量中。因此，如果 TS 是 FTTIMEOUTS 结构并且要写入的字节数为 dwToWrite，则写入超时 wrTO 计算如下。

wrTO = (dwToWrite \* TS.WriteTotalTimeoutMultiplier) + TS.WriteTotalTimeoutConstant

Linux 和 Mac OS X 当前忽略 ReadIntervalTimeout、ReadTotalTimeoutMultiplier 和 WriteTotalTimeoutMultiplier。

### 例子

FT\_HANDLE ftHandle； // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FT 超时 ftTS；

ftTS.ReadIntervalTimeout = 0;

ftTS.ReadTotalTimeoutMultiplier = 0;

ftTS.ReadTotalTimeoutConstant = 100;

ftTS.WriteTotalTimeoutMultiplier = 0;

ftTS.WriteTotalTimeoutConstant = 200;

if (FT\_W32\_SetCommTimeouts(ftHandle,&ftTS))

; // FT\_W32\_SetCommTimeouts 确定

别的

; // FT\_W32\_SetCommTimeouts 失败

## FT\_W32\_GetCommTimeouts

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数获取指定设备当前的读写请求超时参数。

### 定义

BOOL **FT\_W32\_GetCommTimeouts** (FT\_HANDLE *ftHandle* , LPFTTIMEOUTS *lpftTimeouts* )

### 参数

*ftHandle* 的句柄 设备。

*lpftTimeouts* 指向 FTTIMEOUTS 结构的指针，用于存储超时 信息。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

有关如何使用超时的说明，请参阅[FT\_W32\_SetCommTimeouts](#_bookmark79) 。

### 例子

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); FT 超时 ftTS；

if (FT\_W32\_GetCommTimeouts(ftHandle,&ftTS))

; // FT\_W32\_GetCommTimeouts 确定

别的

; // FT\_W32\_GetCommTimeouts 失败

## FT\_W32\_SetCommBreak

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将通信线路置于 BREAK 状态。

### 定义

BOOL **FT\_W32\_SetCommBreak** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

if (!FT\_W32\_SetCommBreak(ftHandle))

; // FT\_W32\_SetCommBreak 失败

别的

; // FT\_W32\_SetCommBreak 确定

## FT\_W32\_ClearCommBreak

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

将通信线路置于非 BREAK 状态。

### 定义

BOOL **FT\_W32\_ClearCommBreak** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

如果（！FT\_W32\_ClearCommBreak（ftHandle））{

// FT\_W32\_ClearCommBreak 失败

}

别的{

}

// FT\_W32\_ClearCommBreak 确定

## FT\_W32\_SetCommMask

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数指定设备必须监视的事件。

### 定义

BOOL **FT\_W32\_SetCommMask** (FT\_HANDLE *ftHandle* , DWORD *dwMask* )

### 参数

*ftHandle* 的句柄 设备。

*dwMask* 包含设备必须监视的事件的掩码。 这 可以是以下组合 下列的：

EV\_BREAK – 检测到 BREAK 条件 EV\_CTS – 清除发送 (CTS) 变化 EV\_DSR – 数据集就绪 (DSR) 变化 EV\_ERR – 线路状态错误

EV\_RING – 环指示符 (RI) 的变化

EV\_RLSD – 接收线路信号变化检测 (RLSD) EV\_RXCHAR – 接收到的字符

EV\_RXFLAG – 收到事件字符 EV\_TXEMPTY – 发送器为空

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

该函数指定设备应监视的事件。应用程序可以调用该函数

[FT\_W32\_WaitCommEvent](#_bookmark85)等待事件发生。

### 例子

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置 FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwMask = EV\_CTS | EV\_DSR；

if (!FT\_W32\_SetCommMask(ftHandle,dwMask))

; // FT\_W32\_SetCommMask 失败

别的

; // FT\_W32\_SetCommMask 确定

## FT\_W32\_GetCommMask

**支持的操作系统**Windows（2000 及更高版本）**摘要**

检索设备当前正在监视的事件。

### 定义

BOOL **FT\_W32\_GetCommMask** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwEventMask* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwEventMask* 指向接收掩码的位置的指针，该掩码包含 事件

目前已启用。该参数可以是以下一个或多个值：

EV\_BREAK – 检测到 BREAK 条件 EV\_CTS – 清除发送 (CTS) 变化 EV\_DSR – 数据集就绪 (DSR) 变化

### 返回 价值

EV\_ERR – 线路状态错误

EV\_RING – 环指示符 (RI) 的变化

EV\_RLSD – 接收线路信号变化检测 (RLSD) EV\_RXCHAR – 接收到的字符

EV\_RXFLAG – 收到事件字符 EV\_TXEMPTY – 发送器为空

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

该函数返回设备当前正在监控的事件。对这些事件的事件监控

[FT\_W32\_SetCommMask函数](#_bookmark83)启用。

### 例子

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); DWORD dwMask；

if (!FT\_W32\_GetCommMask(ftHandle,&dwMask))

; // FT\_W32\_GetCommMask 失败

别的

; // FT\_W32\_GetCommMask 确定

## FT\_W32\_WaitCommEvent

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

该函数等待事件发生。

### 定义

BOOL **FT\_W32\_WaitCommEvent** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwEvent* ,

LPOVERLAPPED *lp重叠*)

### 参数

*ftHandle* 的句柄 设备。

*lpdwEvent* 指向接收掩码的位置的指针，该掩码包含 这 事件 发生。

*lpOverlapped* 指向重叠的指针 结构。

### 返回值

如果函数成功，则返回值非零。

如果函数不成功，则返回值为零。

### 评论

该函数支持非重叠和重叠 I/O，但在 Windows CE 和 Linux 下仅支持非重叠 IO。

非重叠 I/O

对于非重叠 I/O ，参数*lpOverlapped必须为 NULL。*

[FT\_W32\_SetCommMask](#_bookmark83)中指定的事件之前，此函数不会返回 已经发生了。发生并导致该函数返回的事件存储在*lpdwEvent中*。

重叠 I/O

当设备已打开以进行重叠 I/O 时，应用程序可以发出请求并在请求挂起时执行一些附加工作。这与非重叠 I/O 的情况形成对比，在非重叠 I/O 中，应用程序发出请求并仅在请求完成后才再次接收控制。

参数*lpOverlapped*必须指向已初始化的 OVERLAPPED 结构。

[FT\_W32\_SetCommMask](#_bookmark83)中指定的事件之前，此函数不会返回 已经发生了。

如果事件已经发生，则请求立即完成，并且返回代码非零。发生的事件存储在*lpdwEvent中*。

如果事件尚未发生，则请求立即完成，并且返回代码为零，表示发生错误。应用程序应调用[FT\_W32\_GetLastError](#_bookmark87)来获取错误原因。如果错误代码为ERROR\_IO\_PENDING，则重叠操作仍在进行中，应用程序可以执行其他处理。最终，应用程序通过调用[FT\_W32\_GetOverlappedResult](#_bookmark73)检查重叠请求的结果。发生并导致该函数返回的事件存储在*lpdwEvent中*。

### 例子

1. 此示例显示如何使用非重叠向设备写入 128 字节 输入/输出。

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置非重叠 I/O FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwEvents；

if (FT\_W32\_WaitCommEvent(ftHandle, &dwEvents, NULL))

; // FT\_W32\_WaitCommEvents 确定

别的

; // FT\_W32\_WaitCommEvents 失败

1. 此示例显示如何使用重叠将 128 字节写入设备 输入/输出。

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置重叠 I/O FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

DWORD dwEvents；

双字 dwRes；

重叠的 osWait = { 0 };

if (!FT\_W32\_WaitCommEvent(ftHandle, &dwEvents, &osWait)) {

如果（FT\_W32\_GetLastError（ftHandle）== ERROR\_IO\_PENDING）{

// 等待被延迟，所以做一些其他的事情直到...

if (!FT\_W32\_GetOverlappedResult(ftHandle, &osWait, &dwRes, FALSE))

; // 错误

别的

}

}

别的 {

; // FT\_W32\_WaitCommEvent 确定

// 发生的事件存储在 dwEvents 中

// FT\_W32\_WaitCommEvent 确定

// 发生的事件存储在 dwEvents 中

}

## FT\_W32\_PurgeComm

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

此功能会清除设备。

### 定义

BOOL **FT\_W32\_PurgeComm** (FT\_HANDLE *ftHandle* , DWORD *dwFlags* )

### 参数

*ftHandle* 的句柄 设备。

*dwFlags* 指定要采取的操作。该动作可以是 组合 的 下列的：

PURGE\_TXABORT – 终止未完成的重叠写入

PURGE\_RXABORT – 终止未完成的重叠读取

PURGE\_TXCLEAR – 清除发送缓冲区 PURGE\_RXCLEAR – 清除接收缓冲区

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**例子**

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle);

如果（FT\_W32\_PurgeComm（ftHandle，PURGE\_TXCLEAR | PURGE\_RXCLEAR））

; // FT\_W32\_PurgeComm 确定

别的

; // FT\_W32\_PurgeComm 失败

## FT\_W32\_GetLastError

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取设备上发生的最后一个错误。

### 定义

DWORD **FT\_W32\_GetLastError** (FT\_HANDLE *ftHandle* )

### 参数

*ftHandle* 的句柄 设备。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。**评论**

此函数通常与重叠 I/O 一起使用，因此在 Windows CE 中不受支持。为一个

其使用说明，请参阅[FT\_W32\_ReadFile](#_bookmark71)和[FT\_W32\_WriteFile](#_bookmark72) 。

在 Linux 和 Mac OS X 中，此函数返回直接映射到 FT 错误的 DWORD（例如 FT\_INVALID\_HANDLE 错误号）。

## FT\_W32\_ClearCommError

### 支持的操作系统

Linux

Mac OS X（10.4 及更高版本） Windows（2000 及更高版本） Windows CE（4.2 及更高版本）**摘要**

获取有关通信错误的信息并获取设备的当前状态。

### 定义

BOOL **FT\_W32\_ClearCommError** (FT\_HANDLE *ftHandle* , LPDWORD *lpdwErrors,*

LPFTCOMSTAT *lpftComstat* )

### 参数

*ftHandle* 的句柄 设备。

*lpdwErrors* 包含错误的变量 面具。

*lpftComstat* 指向 FTCOMSTAT 的指针 结构。

### 返回值

如果函数成功，则返回值非零。如果函数不成功，则返回值为零。

### 例子

静态 COMSTAT oldCS = {0}；静态 DWORD dwOldErrors = 0;

FT\_HANDLE ftHandle; // 通过 FT\_W32\_CreateFile 设置

FT\_STATUS ftStatus = FT\_Open(0, &ftHandle); COMSTAT 新CS；

DWORD dwErrors；

BOOL bChanged = FALSE;

if (!FT\_W32\_ClearCommError(ftHandle, &dwErrors, (FTCOMSTAT \*)&新CS))

; // FT\_W32\_ClearCommError 失败

if (dwErrors != dwOldErrors) { bChanged = TRUE; dwOldErrors = dwErrors;

}

if (memcmp(&oldCS, &newCS, sizeof(FTCOMSTAT))) { bChanged = TRUE;

旧CS = 新CS；

}

如果（b改变）{

if (dwErrors & CE\_BREAK)

; // 检测到 BREAK 条件 if (dwErrors & CE\_FRAME)

; // 检测到帧错误 if (dwErrors & CE\_RXOVER)

; // 接收缓冲区溢出 if (dwErrors & CE\_TXFULL)

; // 发送缓冲区已满 if (dwErrors & CE\_OVERRUN)

; // 字符缓冲区溢出 if (dwErrors & CE\_RXPARITY)

; // 检测到奇偶校验错误 if (newCS.fCtsHold)

; // 发射机等待 CTS if (newCS.fDsrHold)

; // 发射机正在等待 DSR if (newCS.fRlsdHold)

; // 发射机正在等待 RLSD if (newCS.fXoffHold)

; // 发送器正在等待，因为收到了 XOFF if (newCS.fXoffSent)

; //

if (newCS.fEof)

; // 已收到文件结束字符 if (newCS.fTxim)

; // Tx 立即字符排队等待传输

// newCS.cbInQue 包含接收队列中的字节数

// newCS.cbOutQue 包含传输队列中的字节数

}

# 接触 信息

|  |  |  |  |
| --- | --- | --- | --- |
| 总部 – 格拉斯哥, 英国 |  | 分支机构 – 美国俄勒冈州泰格德 | |
| 未来科技设备国际有限公司 Unit 1, 2 Seaward Place, Centurion Business Park Glasgow G41 1HH  英国  电话：+44 (0) 141 429 2777  传真：+44 (0) 141 429 2758 | | 未来科技设备国际有限公司（美国）7130 SW Fir Loop  泰格德，OR 97223-8160 美国  电话：+1 (503) 547 0988  传真：+1 (503) 547 0987 | |
| 电子邮件（销售） | [sales1@ftdichip.com](mailto:sales1@ftdichip.com) | 电子邮件（销售） | [us.sales@ftdichip.com](mailto:us.sales@ftdichip.com) |
| 电子邮件（支持） | [support1@ftdichip.com](mailto:support1@ftdichip.com) | 电子邮件（支持） | [us.support@ftdichip.com](mailto:us.support@ftdichip.com) |
| 电子邮件（一般查询） | [admin1@ftdichip.com](mailto:admin1@ftdichip.com) | 电子邮件（一般查询） | [us.admin@ftdichip.com](mailto:us.admin@ftdichip.com) |

|  |  |  |  |
| --- | --- | --- | --- |
| 分公司 – 台湾台北 | | 分公司 – 中国上海 | |
| 未来科技设备国际有限公司 (中国台湾地区)  二段516号2楼台北市内湖路1号 114  中华民国台湾  电话：+886 (0) 2 8797 1330  传真：+886 (0) 2 8791 3576 | | 未来科技设备国际有限公司（中国）淮海西路666号1103室  中国上海 200052  电话：+86 21 62351596  传真：+86 21 62351595 | |
| 电子邮件（销售） | [tw.sales1@ftdichip.com](mailto:tw.sales1@ftdichip.com) | 电子邮件（销售） | [cn.sales@ftdichip.com](mailto:cn.sales@ftdichip.com) |
| 电子邮件（支持） | [tw.support1@ftdichip.com](mailto:tw.support1@ftdichip.com) | 电子邮件（支持） | [cn.support@ftdichip.com](mailto:cn.support@ftdichip.com) |
| 电子邮件（一般查询） | [tw.admin1@ftdichip.com](mailto:tw.admin1@ftdichip.com) | 电子邮件（一般查询） | [cn.admin@ftdichip.com](mailto:cn.admin@ftdichip.com) |

#### 网站

[http://ftdichip.com](http://ftdichip.com/)

#### 经销商和销售代表

[FTDI 网站](http://www.ftdichip.com/)的销售网络页面，了解我们在您所在国家/地区的经销商和销售代表的联系方式。

系统和设备制造商和设计人员有责任确保其系统以及其系统中包含的任何 Future Technology Devices International Ltd (FTDI) 设备满足所有适用的安全、法规和系统级性能要求。本文档中所有与应用相关的信息（包括应用描述、建议的 FTDI 设备和其他材料）仅供参考。虽然 FTDI 已尽力确保其准确性，但该信息仍需客户确认，并且 FTDI 对系统设计和 FTDI 提供的任何应用程序协助不承担任何责任。在生命支持和/或安全应用中使用 FTDI 设备的风险完全由用户自行承担，并且用户同意为 FTDI 辩护、赔偿并使 FTDI 免受因此类使用而造成的任何及所有损害、索赔、诉讼或费用。本文如有更改，恕不另行通知。本文档的发布并不暗示可以自由使用专利或其他知识产权。未经版权所有者事先书面同意，不得以任何材料或电子形式改编或复制本文档中包含的全部或任何信息或描述的产品。未来技术设备国际有限公司，Unit 1, 2 Seaward Place, Centurion Business Park, Glasgow G41 1HH, United Kingdom。苏格兰注册公司编号：SC136640

# 附录 A - 类型定义

UCHAR 无符号字符 (1 字节）PUCHAR 指针 未签名 char PCHAR 指向的指针 字符

DWORD 无符号长整型 (4 字节） LPDWORD 指向 unsigned long FT\_HANDLE 的指针 处理

FT\_STATUS（双字）

FT\_OK = 0

FT\_INVALID\_HANDLE = 1

FT\_DEVICE\_NOT\_FOUND = 2

FT\_DEVICE\_NOT\_OPENED = 3

FT\_IO\_错误 = 4

FT\_INSUFFICIENT\_RESOURCES = 5

FT\_INVALID\_PARAMETER = 6

FT\_INVALID\_BAUD\_RATE = 7

FT\_DEVICE\_NOT\_OPENED\_FOR\_ERASE = 8

FT\_DEVICE\_NOT\_OPENED\_FOR\_WRITE = 9

FT\_FAILED\_TO\_WRITE\_DEVICE = 10

FT\_EEPROM\_READ\_FAILED = 11

FT\_EEPROM\_WRITE\_FAILED = 12

FT\_EEPROM\_ERASE\_FAILED = 13

FT\_EEPROM\_NOT\_PRESENT = 14

FT\_EEPROM\_NOT\_PROGRAMMED = 15

FT\_INVALID\_ARGS = 16

FT\_NOT\_SUPPORTED = 17

FT\_OTHER\_错误 = 18

标志（参见[FT \_ListDevices](#_bookmark9) ） FT\_LIST\_NUMBER\_ONLY = 0x80000000 FT\_LIST\_BY\_INDEX = 0x40000000 FT\_LIST\_ALL = 0x20000000

标志（参见[FT \_OpenEx](#_bookmark11) ）

FT\_OPEN\_BY\_SERIAL\_NUMBER = 1

FT\_OPEN\_BY\_DESCRIPTION = 2

FT\_OPEN\_BY\_LOCATION = 4

FT\_DEVICE（双字）

FT\_DEVICE\_232BM = 0

FT\_DEVICE\_232AM = 1

FT\_DEVICE\_100AX = 2

FT\_DEVICE\_UNKNOWN = 3

FT\_DEVICE\_2232C = 4

FT\_DEVICE\_232R = 5

FT\_DEVICE\_2232H = 6

FT\_DEVICE\_4232H = 7

FT\_DEVICE\_232H = 8

FT\_DEVICE\_X\_SERIES = 9

驱动程序类型 FT\_DRIVER\_TYPE\_D2XX 0

FT\_DRIVER\_TYPE\_VCP 1

字长（参见[FT \_SetDataCharacteristics](#_bookmark17) ） FT\_BITS\_8 = 8

FT\_BITS\_7 = 7

停止位（[参见 FT \_SetDataCharacteristics](#_bookmark17) ） FT\_STOP\_BITS\_1 = 0

FT\_STOP\_BITS\_2 = 2

奇偶校验（参见[FT \_SetDataCharacteristics](#_bookmark17) ） FT\_PARITY\_NONE = 0

FT\_PARITY\_ODD = 1

FT\_PARITY\_EVEN = 2

FT\_PARITY\_MARK = 3

FT\_PARITY\_SPACE = 4

流量控制（参见[FT \_SetFlowControl](#_bookmark19) ） FT\_FLOW\_NONE = 0x0000 FT\_FLOW\_RTS\_CTS = 0x0100 FT\_FLOW\_DTR\_DSR = 0x0200 FT\_FLOW\_XON\_XOFF = 0x0400

清除 RX 和 TX 缓冲区（参见[FT \_Purge](#_bookmark36) ） FT\_PURGE\_RX = 1

FT\_PURGE\_TX = 2

通知事件（参见[FT \_SetEventNotification](#_bookmark32) ） FT\_EVENT\_RXCHAR = 1

FT\_EVENT\_MODEM\_STATUS = 2

FT\_EVENT\_LINE\_STATUS = 4

调制解调器状态（请参阅

[FT\_GetModemStatus](#_bookmark25) )

CTS = 0x10 DSR = 0x20 RI = 0x40 DCD = 0x80

线路状态（参见

[FT\_GetModemStatus](#_bookmark25) )

OE = 0x02 PE = 0x04 FE = 0x08 BI = 0x10

位模式（参见[FT \_SetBitMode](#_bookmark65) ） FT\_BITMODE\_RESET = 0x00 FT\_BITMODE\_ASYNC\_BITBANG = 0x01 FT\_BITMODE\_MPSSE = 0x02

FT\_BITMODE\_SYNC\_BITBANG = 0x04 FT\_BITMODE\_MCU\_HOST = 0x08 FT\_BITMODE\_FAST\_SERIAL = 0x10 FT\_BITMODE\_CBUS\_BITBANG = 0x20 FT\_BITMODE\_SYNC\_FIFO = 0x40

FT232R CBUS EEPROM 选项 - FT245R 忽略（参见[FT \_EE\_Program](#_bookmark55)和[FT \_EE\_Read](#_bookmark53) ） FT\_232R\_CBUS\_TXDEN = 0x00

FT\_232R\_CBUS\_PWRON = 0x01 FT\_232R\_CBUS\_RXLED = 0x02 FT\_232R\_CBUS\_TXLED = 0x03 FT\_232R\_CBUS\_TXRXLED = 0x04 FT\_232R\_CBUS\_SLEEP = 0x05 FT\_232R\_CBUS\_CLK48 = 0x06 FT\_232 R\_CBUS\_CLK24 = 0x07 FT\_232R\_CBUS\_CLK12 = 0x08 FT\_232R\_CBUS\_CLK6 = 0x09 FT\_232R\_CBUS\_IOMODE = 0x0A FT\_232R\_CBUS\_BITBANG\_WR = 0x0B FT\_232R\_CBUS\_BITBANG\_RD = 0x0C

FT232H CBUS EEPROM 选项（参见[FT \_EE\_Program](#_bookmark55)和[FT \_EE\_Read](#_bookmark53) ） FT\_232H\_CBUS\_TRISTATE = 0x00

FT\_232H\_CBUS\_RXLED = 0x01 FT\_232H\_CBUS\_TXLED = 0x02 FT\_232H\_CBUS\_TXRXLED = 0x03 FT\_232H\_CBUS\_PWREN = 0x04 FT\_232H\_CBUS\_SLEEP = 0x05 FT\_232H\_CBUS\_DRIVE\_0 = 0x06 FT\_23 2H\_CBUS\_DRIVE\_1 = 0x07 FT\_232H\_CBUS\_IOMODE = 0x08 FT\_232H\_CBUS\_TXDEN = 0x09 FT\_232H\_CBUS\_CLK30 = 0x0A FT\_232H\_CBUS\_CLK15 = 0x0B FT\_232H\_CBUS\_CLK7\_5 = 0x0C

FT X 系列 CBUS 选项 EEPROM 值（请参阅 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program） FT\_X\_SERIES\_CBUS\_TRISTATE = 0x00

FT\_X\_SERIES\_CBUS\_RXLED = 0x01 FT\_X\_SERIES\_CBUS\_TXLED = 0x02 FT\_X\_SERIES\_CBUS\_TXRXLED = 0x03 FT\_X\_SERIES\_CBUS\_PWREN = 0x04 FT\_X\_SERIES\_CBUS\_SLEEP = 0x05 FT\_X\_SERIES\_CBUS\_DRIVE\_0 = 0x0 6 FT\_X\_SERIES\_CBUS\_DRIVE\_1 = 0x07 FT\_X\_SERIES\_CBUS\_IOMODE = 0x08 FT\_X\_SERIES\_CBUS\_TXDEN = 0x09 FT\_X\_SERIES\_CBUS\_CLK24 = 0x0A FT\_X\_SERIES\_CBUS\_CLK12 = 0x0B FT\_X\_SERIES\_CBUS\_CLK6 = 0x0 C

FT\_X\_SERIES\_CBUS\_BCD\_CHARGER = 0x0D FT\_X\_SERIES\_CBUS\_BCD\_CHARGER\_N = 0x0E FT\_X\_SERIES\_CBUS\_I2C\_TXE = 0x0F FT\_X\_SERIES\_CBUS\_I2C\_RXF = 0x10 FT\_X\_SERIES\_CBUS\_VBUS\_SENSE = 0x11 FT \_X\_SERIES\_CBUS\_BITBANG\_WR = 0x12 FT\_X\_SERIES\_CBUS\_BITBANG\_RD = 0x13 FT\_X\_SERIES\_CBUS\_TIMESTAMP = 0x14 FT\_X\_SERIES\_CBUS\_KEEP\_AWAKE = 0x15

FT\_DEVICE\_LIST\_INFO\_NODE（参见[FT \_GetDeviceInfoList](#_bookmark7)和[FT \_GetDeviceInfoDetail](#_bookmark8) ） typedef struct \_ft\_device\_list\_info\_node {

双字标志；

双字类型；

双字 ID；

DWORD 本地 ID；

字符序列号[16]；字符描述[64]； FT\_HANDLE ftHandle;

FT\_DEVICE\_LIST\_INFO\_NODE;

FT\_FLAGS（参见[FT\_DEVICE\_LIST\_INFO\_NODE](#_bookmark102) ） FT\_FLAGS\_OPENED = 0x00000001

FT\_PROGRAM\_DATA\_STRUCTURE

typedef 结构 ft\_program\_data {

双字 签名1； // 标头 - 必须 是 0x0000000 双字 签名2； // 标头 - 必须是 0xffffffff 0xffffffff

双字 版本; // 标头 - FT\_PROGRAM\_DATA 版本

// 0 = 原始值 (FT232B)

// 1 = FT2232 扩展

// 2 = FT232R 扩展

// 3 = FT2232H 扩展

// 4 = FT4232H 扩展

// 5 = FT232H 扩展

单词 供应商 ID； // 0x0403

单词 产品编号； // 0x6001

字符 \*制造商; // “FTDI”

字符 \*制造商ID； // 《金融时报》

字符 \*描述; //“USB HS 串行 转换器”字符 \*序列号; //“FT000001”如果已修复， 或者 空字 最大功率; // 0 < 最大功率 <= 500

单词 即插即用； // 0 = 禁用, 1 = 已启用

单词 自供电； // 0 = 总线供电，1 = 自己 动力字 远程唤醒； // 0 = 没有能力，1 = 有能力的

//

// Rev4 (FT232B) 扩展

//

乌查尔 修订版4； // 如果 Rev4 芯片非零，则为零 否则

乌查尔 异同； // 如果端点是非零 等时的

乌查尔 隔离输出； // 如果输出端点非零 是 等时UCHAR 下拉启用； // 如果下拉则非零 已启用

乌查尔 SerNumEnable; // 如果序列号为非零 用过的

乌查尔 USB版本启用； // 如果芯片非零 用途 USB版本 WORD USB版本； // BCD (0x0200 => USB2)

//

// Rev 5 (FT2232) 扩展

//

乌查尔 修订版5； // 如果 Rev5 芯片非零，则为零 否则

乌查尔 异InA； // 如果端点是非零 等时的

乌查尔 异InB； // 如果端点是非零 等时的

乌查尔 异出A； // 如果输出端点为非零 等时的

乌查尔 异出B； // 如果输出端点非零 是 等时UCHAR 下拉启用5； // 如果下拉则非零 已启用

乌查尔 SerNumEnable5; // 如果序列号为非零 是 使用UCHAR USB版本启用5； // 如果芯片非零 用途 USB版本 WORD USB版本5； // BCD (0x0200 => USB2)

乌查尔 AI 为高电流； // 如果接口为非零 高的 当前的UCHAR BIs高电流； // 如果接口为非零 高的 当前的UCHAR IFAsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 IFIsFifoTar； // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IFAIsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 AIsVCP； // 如果接口要使用 VCP，则非零 司机

乌查尔 IFBIsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 IFBIsFifoTar； // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IFBIsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 BIsVCP； // 如果接口要使用 VCP，则非零 司机

//

// Rev 6 (FT232R) 扩展

//

乌查尔 使用ExtOsc； // 使用 外部的 振荡器 UCHAR HighDriveIO； // 高的 驾驶 I/O UCHAR 端点大小； // 端点 尺寸

乌查尔 下拉启用R； // 如果拉则非零 向下 启用UCHAR SerNumEnableR; // 如果序列号为非零 是 使用UCHAR 反转TXD； // 如果反转则非零 TXD

乌查尔 反转RXD； // 如果反转则非零 接收数据

乌查尔 反转RTS； // 如果反转则非零 实时传输系统

乌查尔 反转CTS； // 如果反转则非零 CTS

乌查尔 反转DTR； // 如果反转则非零 数字TR

乌查尔 反转DSR； // 如果反转则非零 数字SR

乌查尔 反转DCD； // 如果反转则非零 二氯二苯醚

乌查尔 反转RI； // 如果反转则非零 RI

乌查尔 C总线0； // Cbus 复用器 控制

乌查尔 C总线1； // Cbus 复用器 控制

乌查尔 C总线2； // Cbus 复用器 控制

乌查尔 C总线3； // Cbus 复用器 控制

乌查尔 C总线4； // Cbus 复用器 控制

乌查尔 RIsD2XX； // 如果使用 D2XX，则非零 司机

//

// Rev 7 (FT2232H) 扩展

//

乌查尔 下拉启用7； // 如果拉则非零 向下 启用UCHAR SerNumEnable7; // 如果序列号为非零 是 使用UCHAR ALSlowSlew； // 如果 AL 引脚有则非零 慢的 杀死UCHAR AL施密特输入； // 如果 AL 引脚是施密特引脚，则非零 输入

乌查尔 AL驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR AHSlowSlew； // 如果 AH 引脚有则非零 慢的 杀死UCHAR AHS施密特输入； // 如果 AH 引脚为非零 施密特 输入 UCHAR AH驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR BLSlowSlew； // 如果 BL 引脚有则非零 慢的 杀死UCHAR BL施密特输入； // 如果 BL 引脚为非零 施密特 输入 UCHAR BL驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

设计

乌查尔 BHS慢速； // 如果 BH 引脚有则非零 慢的 杀死UCHAR BHS 施密特输入； // 如果 BH 引脚为非零 施密特 输入 UCHAR BH驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR IFAIsFifo7; // 如果接口为 245，则非零 先进先出

乌查尔 IFAIsFifoTar7; // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IFAIsFastSer7； // 如果接口速度快则非零 连续剧

乌查尔 AIsVCP7； // 如果要使用接口则非零 VCP 驱动程序 UCHAR IFBIsFifo7； // 如果接口为 245，则非零 先进先出

乌查尔 IFBIsFifoTar7； // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IFBIsFastSer7； // 如果接口速度快则非零 连续剧

乌查尔 BISVCP7； // 如果接口要使用 VCP，则非零 司机

乌查尔 省电启用； // 如果使用 BCBUS7 来节省电量，则非零 自供电

//

// Rev 8 (FT4232H) 扩展

//

乌查尔 下拉启用8； // 如果拉则非零 向下 启用UCHAR SerNumEnable8; // 如果序列号为非零 是 使用UCHAR AS慢速转换； // 如果 AL 引脚有则非零 慢的 杀死UCHAR 施密特输入； // 如果 AL 引脚是施密特引脚，则非零 输入

乌查尔 A驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR BSlowSlew； // 如果 AH 引脚有则非零 慢的 杀死UCHAR B施密特输入； // 如果 AH 引脚为非零 施密特 输入 UCHAR B驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR CSlowSlew； // 如果 BL 引脚有则非零 慢的 杀死UCHAR 施密特输入； // 如果 BL 引脚为非零 施密特 输入 UCHAR C驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR DS 慢速旋转； // 如果 BH 引脚有则非零 慢的 杀死UCHAR DSchmitt 输入； // 如果 BH 引脚为非零 施密特 输入 UCHAR D驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

乌查尔 ARIIsTXDEN； // 如果端口 A 使用 RI，则非零 RS485 TXD 乌查尔 BRIIsTXDEN； // 如果端口 B 使用 RI，则非零 RS485 TXD 乌查尔 CRIIsTXDEN； // 如果端口 C 使用 RI，则非零 RS485 TXD 乌查尔 DRIIsTXDEN； // 如果端口 D 使用 RI，则非零 RS485 TXD 乌查尔 AIsVCP8； // 如果要使用接口则非零 VCP 驱动程序 UCHAR BISVCP8； // 如果要使用接口则非零 VCP 驱动程序 UCHAR CIsVCP8； // 如果要使用接口则非零 VCP 驱动程序 UCHAR DIsVCP8； // 如果接口要使用 VCP，则非零 司机

//

// Rev 9 (FT232H) 扩展

//

乌查尔 下拉启用H； // 如果拉则非零 向下 启用UCHAR SerNumEnableH； // 如果序列号为非零 是 使用UCHAR ACSlowSlewH； // 如果 AC 引脚有则非零 慢的 杀死UCHAR AC施密特输入H； // 如果 AC 引脚为施密特则非零 输入

乌查尔 交流驱动电流H； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR ADSlowSlewH； // 如果 AD 引脚有则非零 慢的 杀死UCHAR AD施密特输入H； // 如果 AD 引脚为非零 施密特 输入 UCHAR AD驱动电流H； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR Cbus0H； // Cbus 复用器 控制

乌查尔 C总线1H； // Cbus 复用器 控制

乌查尔 Cbus2H； // Cbus 复用器 控制

乌查尔 Cbus3H； // Cbus 复用器 控制

乌查尔 Cbus4H； // Cbus 复用器 控制

乌查尔 C总线5H； // Cbus 复用器 控制

乌查尔 C总线6H； // Cbus 复用器 控制

乌查尔 C总线7H； // Cbus 复用器 控制

乌查尔 Cbus8H； // Cbus 复用器 控制

乌查尔 Cbus9H； // Cbus 复用器 控制

乌查尔 是FifoH； // 如果接口为 245，则非零 先进先出

设计

乌查尔 是FifoTarH； // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IsFastSerH； // 如果接口速度快则非零 连续剧

乌查尔 是FT1248H； // 如果接口为非零 FT1248

乌查尔 FT1248CpolH； // FT1248 时钟极性 - 时钟空闲高 (1) 或时钟空闲 低的 (0) 乌查尔 FT1248LsbH； // FT1248数据为LSB(1)或MSB (0)

乌查尔 FT1248流量控制H； // FT1248流控 使能够

乌查尔 是VCPH； // 如果接口要使用 VCP，则非零 司机

乌查尔 省电启用H； // 如果使用 ACBUS7 来节省电量，则非零 自供电

} FT\_PROGRAM\_DATA, \*PFT\_PROGRAM\_DATA;

EEPROM\_HEADER 结构（参见 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program） typedef struct ft\_eeprom\_header {

FT\_DEVICE 设备类型; // FTxxxx 设备类型为 程序

// 设备描述符选项

单词 供应商 ID； // 0x0403

单词 产品编号； // 0x6001

乌查尔 SerNumEnable; // 如果序列号为非零 用过的

// 配置描述符选项

单词 最大功率; // 0 < 最大功率 <= 500

乌查尔 自供电； // 0 = 总线供电，1 = 自供电 供电的

乌查尔 远程唤醒； // 0 = 没有能力，1 = 有能力的

// 硬件选项

乌查尔 下拉启用； // 如果在挂起时下拉则非零 已启用

} FT\_EEPROM\_HEADER, \*PFT\_EEPROM\_HEADER;

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT232B EEPROM 结构 typedef struct ft\_eeprom\_232b {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

} FT\_EEPROM\_232B，\*PFT\_EEPROM\_232B；

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT2232 EEPROM 结构 typedef struct ft\_eeprom\_2232 {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 AI 为高电流； // 如果接口为高电平则非零 当前的

乌查尔 BIs高电流； // 如果接口为高电平则非零 当前的

// 硬件选项

乌查尔 AIsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 AIsFifoTar； // 如果接口是 245 FIFO CPU，则非零 目标

乌查尔 AIsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 BIsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 BIsFifoTar； // 如果接口是 245 FIFO CPU，则非零 目标

乌查尔 BIsFastSer； // 如果接口速度快则非零 连续剧

// 驱动选项

乌查尔 A驱动程序类型； //

乌查尔 B驱动程序类型； //

} FT\_EEPROM\_2232, \*PFT\_EEPROM\_2232;

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT232R EEPROM 结构 typedef struct ft\_eeprom\_232r {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 是高电流； // 如果接口为高电平则非零 当前的

// 硬件选项

乌查尔 使用ExtOsc； // 使用外部 振荡器

乌查尔 反转TXD； // 如果反转则非零 TXD

乌查尔 反转RXD； // 如果反转则非零 接收数据

乌查尔 反转RTS； // 如果反转则非零 实时传输系统

乌查尔 反转CTS； // 如果反转则非零 CTS

乌查尔 反转DTR； // 如果反转则非零 数字TR

乌查尔 反转DSR； // 如果反转则非零 数字SR

乌查尔 反转DCD； // 非零如果 倒置 DCD UCHAR 反转RI； // 非零如果 倒置 RI 乌查尔 C总线0； // Cbus 复用器 控制

乌查尔 C总线1； // Cbus 复用器 控制

乌查尔 C总线2； // Cbus 复用器 控制

乌查尔 C总线3； // Cbus 复用器 控制

乌查尔 C总线4； // Cbus 复用器 控制

// 驱动选项

乌查尔 驱动程序类型； //

} FT\_EEPROM\_232R，\*PFT\_EEPROM\_232R；

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT2232H EEPROM 结构 typedef struct ft\_eeprom\_2232h {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 ALSlowSlew； // 如果 AL 引脚有则非零 慢的 杀死UCHAR AL施密特输入； // 如果 AL 引脚为非零 施密特 输入 UCHAR AL驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR AHSlowSlew； // 如果 AH 引脚有则非零 慢的 杀死UCHAR AHS施密特输入； // 如果 AH 引脚为非零 施密特 输入 UCHAR AH驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR BLSlowSlew； // 如果 BL 引脚有则非零 慢的 杀死UCHAR BL施密特输入； // 如果 BL 引脚为非零 施密特 输入 UCHAR BL驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR BHS慢速； // 如果 BH 引脚有则非零 慢的 杀死UCHAR BHS 施密特输入； // 如果 BH 引脚为非零 施密特 输入 UCHAR BH驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

// 硬件选项

乌查尔 AIsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 AIsFifoTar； // 如果接口是 245 FIFO CPU，则非零 目标

乌查尔 AIsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 BIsFifo； // 如果接口为 245，则非零 先进先出

乌查尔 BIsFifoTar； // 如果接口是 245 FIFO CPU，则非零 目标

乌查尔 BIsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 省电启用； // 如果使用 BCBUS7 来节省电量，则非零 为了

// 自供电设计

// 驱动选项

乌查尔 A驱动程序类型； //

乌查尔 B驱动程序类型； //

} FT\_EEPROM\_2232H, \*PFT\_EEPROM\_2232H;

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT4232H EEPROM 结构 typedef struct ft\_eeprom\_4232h {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 AS慢速转换； // 如果 A 引脚有则非零 慢的 杀死UCHAR 施密特输入； // 如果 A 引脚是施密特引脚，则非零 输入

乌查尔 A驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR BSlowSlew； // 如果 B 引脚速度较慢则非零 屠杀

乌查尔 B施密特输入； // 如果 B 引脚为非零 施密特 输入 UCHAR B驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR CSlowSlew； // 如果 C 引脚速度较慢则非零 屠杀

乌查尔 施密特输入； // 如果 C 引脚为非零 施密特 输入 UCHAR C驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR DS 慢速旋转； // 如果 D 引脚速度较慢则非零 屠杀

乌查尔 DSchmitt 输入； // 如果 D 引脚为非零 施密特 输入 UCHAR D驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

// 硬件选项

乌查尔 ARIIsTXDEN； // 如果端口 A 使用 RI，则非零 RS485 TXD 乌查尔 BRIIsTXDEN； // 如果端口 B 使用 RI，则非零 RS485 TXD 乌查尔 CRIIsTXDEN； // 如果端口 C 使用 RI，则非零 RS485 TXD 乌查尔 DRIIsTXDEN； // 如果端口 D 使用 RI 作为 RS485，则非零 TXDEN

// 驱动选项

乌查尔 A驱动程序类型； //

乌查尔 B驱动程序类型； //

乌查尔 CDriver类型； //

乌查尔 D驱动程序类型； //

} FT\_EEPROM\_4232H, \*PFT\_EEPROM\_4232H;

// 与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT232H EEPROM 结构 typedef struct ft\_eeprom\_232h {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 ACS 慢速转换； // 如果 AC 总线引脚有则非零 慢的 杀死UCHAR AC施密特输入； // 如果交流总线引脚为非零 施密特 输入 UCHAR 交流驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR ADSlowSlew； // 如果 AD 总线引脚有则非零 慢的 杀死UCHAR AD施密特输入； // 如果 AD 总线引脚为非零 施密特 输入 UCHAR AD驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

// CBUS 选项

乌查尔 C总线0； // Cbus 复用器 控制

乌查尔 C总线1； // Cbus 复用器 控制

乌查尔 C总线2； // Cbus 复用器 控制

乌查尔 C总线3； // Cbus 复用器 控制

乌查尔 C总线4； // Cbus 复用器 控制

乌查尔 C总线5； // Cbus 复用器 控制

低 (0)

乌查尔 C总线6； // Cbus 复用器 控制

乌查尔 C总线7； // Cbus 复用器 控制

乌查尔 C总线8； // Cbus 复用器 控制

乌查尔 C总线9； // Cbus 复用器 控制

// FT1248 选项

乌查尔 FT1248Cpol； // FT1248时钟极性-时钟空闲高(1)或时钟 闲置的

乌查尔 FT1248Lsb； // FT1248数据为LSB(1)或MSB (0)

乌查尔 FT1248流量控制； // FT1248流控 使能够

// 硬件选项

乌查尔 是Fifo； // 如果接口为 245，则非零 先进先出

乌查尔 是FifoTar； // 如果接口是 245 FIFO，则非零 中央处理器 目标UCHAR IsFastSer； // 如果接口速度快则非零 连续剧

乌查尔 IsFT1248 // 非零 if 接口 是 FT1248 乌恰尔 省电启用；

// 驱动程序选项 UCHAR DriverType;

} FT\_EEPROM\_232H, \*PFT\_EEPROM\_232H;

与 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 一起使用的 FT X 系列 EEPROM 结构 typedef struct ft\_eeprom\_x\_series {

// 公共头

FT\_EEPROM\_HEADER common;//所有设备EEPROM的公共元素

// 驱动选项

乌查尔 ACS 慢速转换； // 如果 AC 总线引脚有则非零 慢的 杀死UCHAR AC施密特输入； // 如果交流总线引脚为非零 施密特 输入 UCHAR 交流驱动电流； // 有效值为 4mA、8mA、 12毫安， 16mA UCHAR ADSlowSlew； // 如果 AD 总线引脚有则非零 慢的 杀死UCHAR AD施密特输入； // 如果 AD 总线引脚为非零 施密特 输入 UCHAR AD驱动电流； // 有效值为 4mA、8mA、12mA、 16毫安

// CBUS 选项

乌查尔 C总线0； // Cbus 复用器 控制

乌查尔 C总线1； // Cbus 复用器 控制

乌查尔 C总线2； // Cbus 复用器 控制

乌查尔 C总线3； // Cbus 复用器 控制

乌查尔 C总线4； // Cbus 复用器 控制

乌查尔 C总线5； // Cbus 复用器 控制

乌查尔 C总线6； // Cbus 复用器 控制

// UART信号选项

乌查尔 反转TXD； // 如果反转则非零 TXD

乌查尔 反转RXD； // 如果反转则非零 接收数据

乌查尔 反转RTS； // 如果反转则非零 实时传输系统

乌查尔 反转CTS； // 如果反转则非零 CTS

乌查尔 反转DTR； // 如果反转则非零 数字TR

乌查尔 反转DSR； // 如果反转则非零 数字SR

乌查尔 反转DCD； // 如果反转则非零 二氯二苯醚

乌查尔 反转RI； // 如果反转则非零 RI

// 电池充电检测选项

乌查尔 BCD 启用； // 启用电池充电器 检测

乌查尔 BCDForceCbusPWREN； // 断言电源使能信号打开 中央总线 当充电口 检测到

乌查尔 BCD禁用睡眠； // 强制设备永远不进入睡眠状态 模式

// I2C 选项

单词 I2C从机地址； // I2C从设备 地址

双字 I2C设备ID； // I2C设备 ID

乌查尔 I2C禁用施密特； // 禁用 I2C 施密特 扳机

// FT1248 选项

乌查尔 FT1248Cpol； // FT1248时钟极性-时钟空闲高电平 (1) 或时钟空闲低电平 (0)

乌查尔 FT1248Lsb； // FT1248数据为LSB(1)或MSB (0)

乌查尔 FT1248流量控制； // FT1248流控 使能够

// 硬件选项

UCHAR RS485回声抑制； UCHAR 省电启用；

// 驱动程序选项 UCHAR DriverType;

} FT\_EEPROM\_X\_SERIES, \*PFT\_EEPROM\_X\_SERIES;

Win32

OPEN\_EXISTING = 3 FILE\_ATTRIBUTE\_NORMAL = 0x00000080 FILE\_FLAG\_OVERLAPPED = 0x40000000 GENERIC\_READ = 0x80000000 GENERIC\_WRITE = 0x40000000

重叠结构 typedef struct \_OVERLAPPED {

ULONG\_PTR 内部； ULONG\_PTR 内部高电平；联盟{

结构体{

};

双字偏移量； DWORD 偏移高；

PVOID 指针；

};

处理 h事件；

} 重叠，\*LP重叠；

CLRDTR = 6 – 清除 DTR 信号 CLRRTS = 4 – 清除 RTS 信号 SETDTR = 5 – 设置 DTR 信号 SETRTS = 3 – 设置 RTS 信号 SETBREAK = 8 – 设置 BREAK 状况

CLRBREAK = 9 – 清除 BREAK 条件

MS\_CTS\_ON = 0x0010 – 清除发送 (CTS) 开启 MS\_DSR\_ON = 0x0020 – 数据集就绪 (DSR) 开启 MS\_RING\_ON = 0x0040 – 振铃指示器 (RI) 开启

MS\_RLSD\_ON = 0x0080 – 接收线路信号检测 (RLSD) 开启

FTDCB 结构 typedef struct \_FTDCB {

DWORD DCB 长度； // 大小(FTDCB)

DWORD 波特率； // 运行波特率 DWORD fBinary: 1; // 二进制模式（跳过 EOF 检查） DWORD fParity: 1; // 启用奇偶校验

DWORD fOutxCtsFlow:1; // 输出上的 CTS 握手 DWORD fOutxDsrFlow:1; // 输出上的 DSR 握手 DWORD fDtrControl:2; // DTR 流量控制

DWORD fDsr灵敏度：1； // DSR 灵敏度

DWORD fTXContinueOnXoff: 1; // 当 Xoff 发送 DWORD fOutX: 1 时继续 TX; // 使能输出 X-ON/X-OFF

双字 fInX: 1; // 启用输入 X-ON/X-OFF DWORD fErrorChar: 1; // 启用错误替换 DWORD fNull: 1; // 启用空剥离

DWORD fRtsControl:2; // Rts 流量控制

DWORD fAbortOnError:1; // 出现错误时中止所有读写操作 DWORD fDummy2:17; // 预订的

WORD w保留； // 当前未使用 WORD XonLim; // 发送 X-ON 阈值 WORD XoffLim; // 发送 X-OFF 阈值 BYTE ByteSize; // 位/字节数，7-8

字节奇偶校验； // 0-4=无、奇数、偶数、标记、空格 BYTE StopBits; // 0,2 = 1, 2

char XonChar; // Tx 和 Rx X-ON 字符 char XoffChar; // Tx 和 Rx X-OFF 字符 char ErrorChar; // 错误替换 char char EofChar; // 输入字符结束

char EvtChar； // 接收事件字符 WORD wReserved1; // 充满

} FTDCB，\*LPFTDCB；

FTTIMEOUTS 结构 typedef struct \_FTTIMEOUTS {

DWORD 读取间隔超时； // 读取字符之间的最大时间 DWORD ReadTotalTimeoutMultiplier; // 字符乘数 DWORD ReadTotalTimeoutConstant; // 以毫秒为单位的常量 DWORD WriteTotalTimeoutMultiplier; // 字符乘数 DWORD WriteTotalTimeoutConstant; // 以毫秒为单位的常量

} FTTIMEOUTS，\*LPFTTIMEOUTS；

EV\_BREAK = 0x0040 – 检测到 BREAK 条件 EV\_CTS = 0x0008 – 清除发送 (CTS) 发生变化 EV\_DSR = 0x0010 – 数据集就绪 (DSR) 发生变化 EV\_ERR = 0x0080 – 线路状态错误

EV\_RING = 0x0100 – 环指示符 (RI) 的变化

EV\_RLSD = 0x0020 – 接收线路信号检测 (RLSD) 变化 EV\_RXCHAR = 0x0001 – 接收到字符

EV\_RXFLAG = 0x0002 – 收到事件字符 EV\_TXEMPTY = 0x0004 – 发送器为空

PURGE\_TXABORT = 0x0001 – 终止未完成的重叠写入 PURGE\_RXABORT = 0x0002 – 终止未完成的重叠读取 PURGE\_TXCLEAR = 0x0004 – 清除发送缓冲区 PURGE\_RXCLEAR = 0x0008 – 清除接收缓冲区

FTCOMSTAT 结构 typedef struct \_FTCOMSTAT {

双字 fCtsHold : 1;双字 fDsrHold : 1;双字 fRlsdHold : 1; DWORD fXoffHold : 1; DWORD fXoffSent : 1;双字 fEof : 1;

双字 fTxim : 1; DWORD f保留：25； DWORD cbInQue； DWORD cbOutQue；

} FTCOMSTAT, \*LPFTCOMSTAT;

# 附录 B – 参考文献

## 文档参考

不适用

## 缩略语

|  |  |
| --- | --- |
| 条款 | 描述 |
| 清洁发展机制 | 组合驱动器模型。 Windows 驱动程序包包含 D2XX 和 VCP 驱动程序。 |
| D2XX | FTDI 专有的“直接”驱动程序接口（通过 FTD2XX.DLL） |
| VCP | 虚拟COM端口 |

# 附录 C – 表格和图表列表

## 表格列表

未找到数字表条目。

## 图列表

[图 2.1 Windows CDM 司机 架构](#_bookmark2)[**6**](#_bookmark2)

# 附录 D - 修订历史

文档 标题: D2XX 程序员的 指导文件 参考 编号： FT\_000071

清除 编号： FTDI# 170

产品 页面： https: [//ftdichip.com/product-category/products/ic/](https://ftdichip.com/product-category/products/ic/)

文档 反馈： [发送 反馈](mailto:docufeedback@ftdichip.com)

|  |  |  |
| --- | --- | --- |
| **修订** | **变化** | **日期** |
| 1.00 | 以新格式首次发布  包含 CDM 驱动程序 2.04.06 中的所有功能 | 2008年8月 |
| 1.01 | 包括 FT4232H 和 FT2232H 更新的地址 | 2009年1月 |
| 1.02 | 第 65 页 – 从 MCU 主机仿真和快速光电模式中删除了 FT232R 和 FT245R 参考 | 2010年1月 |
| 1.03 | 更正了第 3.32 节 (FT\_Purge) 更新了联系方式 | 2010-09-08 |
| 1.04 | 在5.3节中添加了245同步FIFO模式代码 | 2010-10-28 |
| 1.1 | 通过重新添加 FT4232H 和 FT2232H 扩展更正了文档之前的编辑错误 | 2010-11-04 |
| 1.2 | 添加了对 FT232H 的参考，包括 EEPROM 格式  许多格式修复  附录中的扩展定义反映了 CDM 2.08.14 头文件中的更新。 | 2011-04-25 |
| 1.3 | 为 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 添加了第 4.11 和 4.12 节  更新了文档末尾的 ftd2xx.h Attachemnet。 | 2012-02-23 |
| 1.4 | 文档模板更改  附录 A 的更改 - 类型定义  添加了有关制造商、制造商 ID、描述和序列号的信息 | 2019-06-24 |
| 1.5 | 所有代码片段均已在启用 Visual Studio 2019 一致性模式的情况下进行更新和测试。  添加了对 FT\_EEPROM\_Read 和 FT\_EEPROM\_Program 的 Linux 支持。  对 FT\_EE\_UASize、FT\_SetBitMode、FT\_GetBitMode 和 FT\_W32\_WaitCommEvent 进行了细微更改。 | 2023年7月9日 |