

D2XX Programmer's Guide Extras

Contents

Introduction

New Functions

FT_GetDeviceInfo

FT_SetResetPipeRetryCount

FT_StopInTask

FT_RestartInTask

FT_ResetPort

Extensions for Location IDs

FT_ListDevices

FT_OpenEx

FT_W32_CreateFile

Introduction

This document contains descriptions of new functions and extensions to existing functions that have not yet been incorporated into the D2XX Programmer's Guide.

New Functions

The functions **FT_GetDeviceInfo**, **FT_SetResetPipeRetryCount**, **FT_StopInTask**, **FT_RestartInTask**, and **FT_ResetPort** have been added to the D2XX API.

FT_GetDeviceInfo

Description

Get device information.

Syntax

FT_STATUS **FT_GetDeviceInfo** (FT_HANDLE ftHandle, FT_DEVICE *pftType, LPDWORD lpdwID, PCHAR pcSerialNumber, PCHAR pcDescription, PVOID pvDummy)

Parameters

<i>ftHandle</i>	FT_HANDLE: handle returned from <i>FT_Open</i> or <i>FT_OpenEx</i> .
<i>pftType</i>	Pointer to unsigned long to store device type.
<i>lpdwId</i>	Pointer to unsigned long to store device ID.
<i>pcSerialNumber</i>	Pointer to buffer to store device serial number as a null-terminated string.
<i>pcDescription</i>	Pointer to buffer to store device description as a null-terminated string.
<i>pvDummy</i>	Reserved for future use - should be set to NULL.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to return the device type, device ID, device description and serial number. The device ID is encoded in a DWORD - the most significant word contains the vendor ID, and the least significant word contains the product ID. So the returned ID 0x04036001 corresponds to the device ID VID_0403&PID_6001.

Example

This example shows how to get information about a device.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_DEVICE ftDevice;
FT_STATUS ftStatus;
DWORD deviceID;
char SerialNumber[16];
char Description[64];

ftStatus = FT_GetDeviceInfo(
    ftHandle,
    &ftDevice,
    &deviceID,
    SerialNumber,
    Description,
    NULL
);
if (ftStatus == FT_OK) {
    if (ftDevice == FT_DEVICE_232BM)
        ; // device is FT232BM
    else if (ftDevice == FT_DEVICE_232R)
        ; // device is FT232R
    else if (ftDevice == FT_DEVICE_232AM)
        ; // device is FT8U232AM
    else if (ftDevice == FT_DEVICE_100AX)
        ; // device is FT8U100AX
    else
        ; // unknown device (this should not happen!)
    // deviceID contains encoded device ID
    // SerialNumber, Description contain 0-terminated strings
}
else {
    // FT_GetDeviceType FAILED!
}
```

FT_SetResetPipeRetryCount

Description

Set the ResetPipeRetryCount.

Syntax

FT_STATUS **FT_SetResetPipeRetryCount** (FT_HANDLE *ftHandle*, DWORD *dwCount*)

Parameters

<i>ftHandle</i>	FT_HANDLE: handle returned from <i>FT_Open</i> or <i>FT_OpenEx</i> .
<i>dwCount</i>	DWORD: unsigned long containing required ResetPipeRetryCount.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to set the ResetPipeRetryCount. ResetPipeRetryCount controls the maximum number of times that the driver tries to reset a pipe on which an error has occurred. ResetPipeRequestRetryCount defaults to 50. It may be necessary to increase this value in noisy environments where a lot of USB errors occur.

Example

This example shows how to set the ResetPipeRetryCount to 100.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;
DWORD dwRetryCount;

dwRetryCount = 100;
ftStatus = FT_SetResetPipeRetryCount(ftHandle,dwRetryCount);
if (ftStatus == FT_OK) {
    // ResetPipeRetryCount set to 100
}
else {
    // FT_SetResetPipeRetryCount FAILED!
}
```

FT_StopInTask

Description

Stops the driver's IN task.

Syntax

```
FT_STATUS FT_StopInTask ( FT_HANDLE ftHandle )
```

Parameters

ftHandle FT_HANDLE: handle returned from *FT_Open* or *FT_OpenEx*.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to put the driver's IN task (read) into a wait state. It can be used in situations where data is being received continuously, so that the device can be purged without more data being received. It is used together with FT_RestartInTask which sets the IN task running again.

Example

This example shows how to use FT_StopInTask.

```

FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

do {
    ftStatus = FT_StopInTask(ftHandle);
} while (ftStatus != FT_OK);

//
// Do something - for example purge device
//

do {
    ftStatus = FT_RestartInTask(ftHandle);
} while (ftStatus != FT_OK);

```

FT_RestartInTask

Description

Restart the driver's IN task.

Syntax

```
FT_STATUS FT_RestartInTask ( FT_HANDLE ftHandle )
```

Parameters

ftHandle FT_HANDLE: handle returned from *FT_Open* or *FT_OpenEx*.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to restart the driver's IN task (read) after it has been stopped by a call to FT_StopInTask.

Example

This example shows how to use FT_RestartInTask.

```

FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

do {
    ftStatus = FT_StopInTask(ftHandle);
} while (ftStatus != FT_OK);

//
// Do something - for example purge device
//

do {
    ftStatus = FT_RestartInTask(ftHandle);
} while (ftStatus != FT_OK);

```

FT_ResetPort

Description

Send a reset command to the port.

Syntax

```
FT_STATUS FT_ResetPort ( FT_HANDLE ftHandle )
```

Parameters

ftHandle FT_HANDLE: handle returned from *FT_Open* or *FT_OpenEx*.

Return Value

FT_OK if successful, otherwise the return value is an FT error code.

Remarks

This function is used to attempt to recover the port after a failure.

Example

This example shows how to reset the port.

```
FT_HANDLE ftHandle;    // valid handle returned from FT_OpenEx
FT_STATUS ftStatus;

ftStatus = FT_ResetPort(ftHandle);
if (ftStatus == FT_OK) {
    // Port has been reset
}
else {
    // FT_ResetPort FAILED!
}
```

Extensions for Location IDs

A location ID is a 32-bit integer (long) that represents the location of the device in the USB tree. For example, if the driver creates a location string for the device of the form a&b&c, the Location ID will be 0x00000abc.

Location IDs can be obtained using the utility USBView that is available from FTDI. To get the Location ID for a specific USB port, run USBView and plug a device into the port that is required. From the menu select Options | LocationIDs. Press F5 to refresh the display, and the port numbers are replaced with Location IDs of the form LocXY where X and Y are hexadecimal digits. XY is used to create a location string of the form X&Y, and the Location ID is 0xAB.

The flag, FT_OPEN_BY_LOCATION, has been introduced to support location IDs and is defined as

```
#define FT_OPEN_BY_LOCATION 4
```

The functions **FT_ListDevices**, **FT_OpenEx**, and **FT_CreateFile** now support location IDs.

FT_ListDevices

Description

Get information concerning the devices currently connected. This function can return information such as the number of devices connected, the device serial number and device description strings, and the location IDs of connected devices.

Syntax

FT_STATUS **FT_ListDevices** (PVOID *pvArg1*,PVOID *pvArg2*,unsigned long *dwFlags*)

Return Value

FT_OK if success, otherwise the return value is an FT error code.

Remarks

This function can be used in a number of ways to return different types of information.

In its simplest form, it can be used to return the number of devices currently connected. If FT_LIST_NUMBER_ONLY bit is set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to a *DWORD* location to store the number of devices currently connected.

It can be used to return device information: if FT_OPEN_BY_SERIAL_NUMBER bit is set in *dwFlags*, the serial number string will be returned; if FT_OPEN_BY_DESCRIPTION bit is set in *dwFlags*, the product description string will be returned; if FT_OPEN_BY_LOCATION bit is set in *dwFlags*, the Location ID will be returned; if none of these bits is set, the serial number string will be returned by default.

It can be used to return device string information for a single device. If FT_LIST_BY_INDEX and FT_OPEN_BY_SERIAL_NUMBER or FT_OPEN_BY_DESCRIPTION bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a buffer to contain the appropriate string. Indexes are zero-based, and the error code FT_DEVICE_NOT_FOUND is returned for an invalid index.

It can be used to return device string information for all connected devices. If FT_LIST_ALL and FT_OPEN_BY_SERIAL_NUMBER or FT_OPEN_BY_DESCRIPTION bits are set in *dwFlags*, the parameter *pvArg1* is interpreted as a pointer to an array of pointers to buffers to contain the appropriate strings, and the parameter *pvArg2* is interpreted as a pointer to a *DWORD* location to store the number of devices currently connected. Note that, for *pvArg1*, the last entry in the array of pointers to buffers should be a NULL pointer so the array will contain one more location than the number of devices connected.

The location ID of a device is returned if FT_LIST_BY_INDEX and FT_OPEN_BY_LOCATION bits are set in *dwFlags*. In this case the parameter *pvArg1* is interpreted as the index of the device, and the parameter *pvArg2* is interpreted as a pointer to a variable of type long to contain the location ID. Indexes are zero-based, and the error code FT_DEVICE_NOT_FOUND is returned for an invalid index.

The location IDs of all connected devices are returned if FT_LIST_ALL and FT_OPEN_BY_LOCATION bits are set in *dwFlags*. In this case, the parameter *pvArg1* is interpreted as a pointer to an array of variable of type long to contain the location IDs, and the parameter *pvArg2* is interpreted as a pointer to a *DWORD* location to store the number of devices currently connected.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
DWORD numDevs;
```

Get the number of devices currently connected

```
ftStatus = FT_ListDevices(&numDevs, NULL, FT_LIST_NUMBER_ONLY);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, number of devices connected is in numDevs  
}  
else {  
    // FT_ListDevices failed  
}
```

Get serial number of first device

```
DWORD devIndex = 0; // first device  
char Buffer[64]; // more than enough room!  
  
ftStatus =  
FT_ListDevices((PVOID)devIndex, Buffer, FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, serial number is in Buffer  
}  
else {  
    // FT_ListDevices failed  
}
```

Note that indexes are zero-based. If more than one device is connected, incrementing devIndex will get the serial number of each connected device in turn.

Get device descriptions of all devices currently connected

```
char *BufPtrs[3]; // pointer to array of 3 pointers  
char Buffer1[64]; // buffer for description of first device  
char Buffer2[64]; // buffer for description of second device  
  
// initialize the array of pointers  
BufPtrs[0] = Buffer1;  
BufPtrs[1] = Buffer2;  
BufPtrs[2] = NULL; // last entry should be NULL  
  
ftStatus = FT_ListDevices(BufPtrs, &numDevs, FT_LIST_ALL|FT_OPEN_BY_DESCRIPTION);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, product descriptions are in Buffer1 and Buffer2, and  
    // numDevs contains the number of devices connected  
}  
else {  
    // FT_ListDevices failed  
}
```

Note that this example assumes that two devices are connected. If more devices are connected, then the size of the array of pointers must be increased and more description buffers allocated.

Get locations of all devices currently connected

```
long locIdBuf[16];  
  
ftStatus = FT_ListDevices(locIdBuf, &numDevs, FT_LIST_ALL|FT_OPEN_BY_LOCATION);  
if (ftStatus == FT_OK) {  
    // FT_ListDevices OK, location IDs are in locIdBuf, and  
    // numDevs contains the number of devices connected  
}  
else {  
    // FT_ListDevices failed  
}
```

Note that this example assumes that no more than 16 devices are connected. If more devices are connected, then the size of the array of pointers must be increased.

FT_OpenEx

Description

Open the specified device and return a handle that will be used for subsequent accesses. The device can be specified by its serial number, device description, or location.

This function can also be used to open multiple devices simultaneously. Multiple devices can be opened at the same time if they can be distinguished by serial number or device description. Alternatively, multiple devices can be opened at the same time using location IDs - location information derived from their physical locations on USB. Location IDs can be obtained using the utility USBView.

Syntax

FT_STATUS **FT_OpenEx** (PVOID *pvArg1*, unsigned long *dwFlags*, FT_HANDLE **ftHandle*)

Return Value

FT_OK if success, otherwise the return value is an FT error code.

Remarks

The meaning of *pvArg1* depends on *dwFlags*: if *dwFlags* is FT_OPEN_BY_SERIAL_NUMBER, *pvArg1* is interpreted as a pointer to a null-terminated string that represents the serial number of the device; if *dwFlags* is FT_OPEN_BY_DESCRIPTION, *pvArg1* is interpreted as a pointer to a null-terminated string that represents the device description; if *dwFlags* is FT_OPEN_BY_LOCATION, *pvArg1* is interpreted as a long value that contains the location ID of the device.

ftHandle is a pointer to a variable of type FT_HANDLE where the handle is to be stored. This handle must be used to access the device.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
FT_STATUS ftStatus2;  
FT_HANDLE ftHandle1;  
FT_HANDLE ftHandle2;  
long dwLoc;
```

Open a device with serial number "FT000001"

```
ftStatus = FT_OpenEx("FT000001", FT_OPEN_BY_SERIAL_NUMBER, &ftHandle1);  
if (ftStatus == FT_OK) {  
    // success - device with serial number "FT000001" is open  
}  
else {  
    // failure  
}
```

Open a device with device description "USB Serial Converter"

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
if (ftStatus == FT_OK) {
    // success - device with device description "USB Serial Converter" is open
}
else {
    // failure
}
```

Open 2 devices with serial numbers "FT000001" and "FT999999"

```
ftStatus = FT_OpenEx("FT000001",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle1);
ftStatus2 = FT_OpenEx("FT999999",FT_OPEN_BY_SERIAL_NUMBER,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
    // success - both devices are open
}
else {
    // failure - one or both of the devices has not been opened
}
```

Open 2 devices with descriptions "USB Serial Converter" and "USB Pump Controller"

```
ftStatus = FT_OpenEx("USB Serial Converter",FT_OPEN_BY_DESCRIPTION,&ftHandle1);
ftStatus2 = FT_OpenEx("USB Pump Controller",FT_OPEN_BY_DESCRIPTION,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
    // success - both devices are open
}
else {
    // failure - one or both of the devices has not been opened
}
```

Open a device at location 23

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
if (ftStatus == FT_OK) {
    // success - device at location 23 is open
}
else {
    // failure
}
```

Open 2 devices at locations 23 and 31

```
dwLoc = 0x23;
ftStatus = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle1);
dwLoc = 0x31;
ftStatus2 = FT_OpenEx(dwLoc,FT_OPEN_BY_LOCATION,&ftHandle2);
if (ftStatus == FT_OK && ftStatus2 == FT_OK) {
    // success - both devices are open
}
else {
    // failure - one or both of the devices has not been opened
}
```

FT_W32_CreateFile

Description

Open the specified device and return a handle which will be used for subsequent accesses. The device can be specified by its serial number, device description, or location.

This function must be used if overlapped I/O is required.

Syntax

FT_HANDLE **FT_W32_CreateFile** (PVOID *pvArg1*, DWORD *dwAccess*, DWORD *dwShareMode*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*, DWORD *dwCreate*, DWORD *dwAttrsAndFlags*, HANDLE *hTemplate*)

Return Value

If the function is successful, the return value is a handle.

If the function is unsuccessful, the return value is the Win32 error code

INVALID_HANDLE_VALUE.

Remarks

The meaning of *pvArg1* depends on *dwAttrsAndFlags*: if FT_OPEN_BY_SERIAL_NUMBER or FT_OPEN_BY_DESCRIPTION is set in *dwAttrsAndFlags*, *pvArg1* contains a pointer to a null terminated string that contains the device's serial number or description; if FT_OPEN_BY_LOCATION is set in *dwAttrsAndFlags*, *pvArg1* is interpreted as a value of type long that contains the location ID of the device.

dwAccess can be GENERIC_READ, GENERIC_WRITE, or both; *dwShareMode* must be set to 0; *lpSecurityAttributes* must be set to NULL; *dwCreate* must be set to OPEN_EXISTING; *dwAttrsAndFlags* is a combination of FILE_ATTRIBUTE_NORMAL, FILE_FLAG_OVERLAPPED if overlapped I/O is used, FT_OPEN_BY_SERIAL_NUMBER or FT_OPEN_BY_DESCRIPTION or FT_OPEN_BY_LOCATION; *hTemplate* must be NULL.

Examples

The examples that follow use these variables.

```
FT_STATUS ftStatus;  
FT_HANDLE ftHandle;  
char Buf[64];
```

Open a device for overlapped I/O using its serial number

```
ftStatus = FT_ListDevices(0,Buf,FT_LIST_BY_INDEX|FT_OPEN_BY_SERIAL_NUMBER);  
ftHandle = FT_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED |  
                             FT_OPEN_BY_SERIAL_NUMBER,  
                             0);  
  
if (ftHandle == INVALID_HANDLE_VALUE)  
    ; // FT_W32_CreateDevice failed
```

Open a device for non-overlapped I/O using its description

```
ftStatus = FT_ListDevices(0,Buf,FT_LIST_BY_INDEX|FT_OPEN_BY_DESCRIPTION);  
ftHandle = FT_W32_CreateFile(Buf,GENERIC_READ|GENERIC_WRITE,0,0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_DESCRIPTION,  
                             0);  
  
if (ftHandle == INVALID_HANDLE_VALUE)  
    ; // FT_W32_CreateDevice failed
```

Open a device for non-overlapped I/O using its location

```
long locID;  
ftStatus = FT_ListDevices(0,&locID,FT_LIST_BY_INDEX|FT_OPEN_BY_LOCATION);  
ftHandle = FT_W32_CreateFile((PVOID) locID,GENERIC_READ|GENERIC_WRITE,0,0,  
                             OPEN_EXISTING,  
                             FILE_ATTRIBUTE_NORMAL | FT_OPEN_BY_LOCATION,  
                             0);  
if (ftHandle == INVALID_HANDLE_VALUE)  
    ; // FT_W32_CreateDevice failed
```