AN1603
APPLICATION NOTE

USING THE ST7 USB DEVICE FIRMWARE UPGRADE DEVELOPMENT KIT (DFU-DK)

by Microcontroller Division Applications

1 INTRODUCTION

This Application Note describes how to use the ST7 USB DFU PC development kit (DFU-DK). It begins with a presentation of the software architecture. It then gives a full description of the DFU-DK programming interface and all the features. Finally, an example is given of a vendor upgrade package developed with the DFU-DK.

The objective of the DFU-DK is to provide you with a simple means of upgrading the firmware of a USB device. It runs under Microsoft Windows and works with any STMicroelectronics USB microcontroller that supports the DFU capability including the ST72F62, ST72F63 and ST72F65x MCU families.

The DFU-DK can be downloaded from ST's website.

Related Documentation:

ST7 USB Device Firmware Upgrade Demonstrator User Manual.

2 LIST OF INSTALLED FILES

This section provides a list of all the DFU-DK files referred to in this application note.

2.1 DEVELOPMENT MODULES FOR BUILDING YOUR OWN PC SOFTWARE

In the **Dev** subdirectory, you will find the following items :

– Bin subdirectory

-ST7DFU.dll : interface dll

-STTubeDevice203.dll : driver dll used by the ST7DFU dll

– Inc subdirectory

-ST7DFU.h : C header containining all prototypes exported by ST7DFU.dll

-USB100.h : Microsoft header for USB.

– Lib subdirectory

-ST7DFU.lib : static lib file

2.2 DOCUMENTATION

In the **Doc** subdirectory, you will find the following items :

– Application note AN1603: this file.

2.3 DFU DRIVER

In the **Driver** subdirectory, you will find the following items :

– STDFU.inf: installation file for the DFU driver that can be customised to meet the requirements of your application

– STTub203.sys: STMicroelectronics tube driver used as a DFU class driver

2.4 EXAMPLE VENDOR UPGRADE SOFTWARE FILES

In the **Sources** subdirectory, you will find the following items :

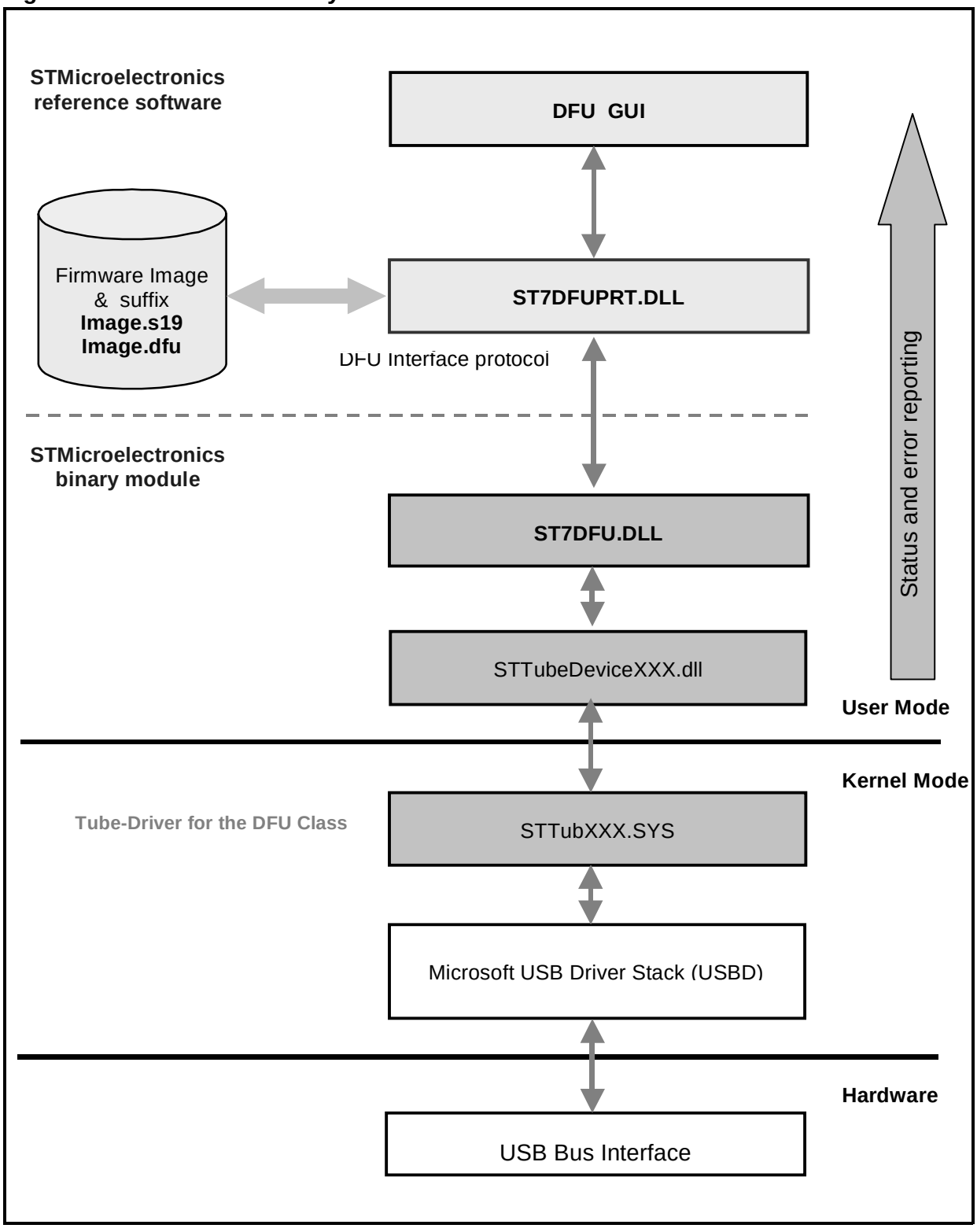
– Binary subdirectory inludes binary modules: debug and release

– DFUDEMO subdirectory includes GUI sources

– ST7DFUPRT subdirectory includes DFU protocol dll sources

3 SOFTWARE ARCHITECTURE

Figure 1. DFU-DK software layers



As shown in Figure 1, we can see that the DFU software is divided into four distinct layers:

– The hardware layer (USB bus interface),

– The kernel layer (Microsoft USB stack and STMicroelectronics tube driver)

and a user application layer that is also divided into two other layers:

– A generic one that manages all DFU requests and is provided as ready binary modules (ST7DFU and STTubeDevice203 dynamic link libraries).

– A vendor-specific one that is provided as an example reference software (protocol management dynamic link library and a demonstration GUI).

This modular architecture was chosen so that the DFU-DK can be used for any application that needs to implement DFU features. Moreover, this will make it easier to add features anytime in the future and manage changes.

Note: The DFU-DK is compatible with Windows 98, 98SE, Me, 2000 and XP and is compliant with the Device Class specification for DFU version1.0 as detailed by the USB-IF.

4 DFU-DK PROGRAMMING INTERFACE

4.1 OVERVIEW

This module is used to allow the application to access the low level USB protocol, such as receiving descriptor information, configuring a device etc, and also for managing all DFU requests that are implemented in this module and are not linked to the application layer.

The following table summarize the DFU class-specific requests used to accomplish the upgrade operation.

Table 1. DFU Class specific requests

bmRequetType	bRequest	wValue	wIndex	wLength	Data
00100001b	DFU_DETACH	wTimeout	Interface	Zero	None
00100001b	DFU_DNLOAD	wBlockNum	Interface	Length	Firmware
10100001b	DFU_UPLOAD	wBlockNum	Interface	Length	Firmware
10100001b	DFU_GETSTATUS	Zero	Interface	6	Status
00100001b	DFU_CLRSTATUS	Zero	Interface	Zero	None
10100001b	DFU_GETSTATE	Zero	Interface	1	State
00100001b	DFU_ABORT	Zero	Interface	Zero	None

4.2 DRIVER INSTALLATION FILE (.INF)

Before creating or customising the DFU inf file, you have to obtain two licence keys for your vendor ID and product IDs, in order to use the DFU driver : One for the product ID of your composite device (Functional mode + DFU mode) and one for the only DFU mode device.

In our example we have two distinct keys, one for the PID=0xDF11 (STMicroelectronics DFU devices) which is “6f72a98be4e1edad” and another one for a composite device, PID= 0xFF03 (STMicroelectronics EvalKit and DFU) which is “d083fc290a34aa18”.

Once you have your licence keys, you need to make your own inf file. The inf file is a text file used by Windows to install a driver. The one given in this package can be customised to fit your needs. Anyway, you have to choose your own name for the inf file, and you shouldn’t use the one given in the package.

After a successful driver installation the correct values in this inf file will be entered in the registery.

4.3 PROGRAMMING INTERFACE

This is the Application-programming interface of the ST7DFU.dll module. The documentation describes the interface in C language.

4.3.1 STDFU_EnumGETNBDevices

4.3.1.1 Prototype

```
DWORD STDFU_EnumGetNbDevices(   PDWORD pNb,
                                HWND hWnd,
                                DWORD Message);
```

4.3.1.2 Description

Browses all DFU devices connected to the PC.

4.3.1.3 Parameters

pNb: pointer to the number of DFU devices

hWnd :window handle

Message: notifies an application of a change to the hardware configuration of a DFU-device.

4.3.1.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code)

4.3.1.5 Remarks

For more details about hWnd and Message parameters please refer to Section 4.3.19

The following section provides all error codes that can be returned by any function:

```
STDFU_ERROR_OFFSET                  0x12340000
STDFU_NOERROR                       STDFU_ERROR_OFFSET
STDFU_MEMORY                        (STDFU_ERROR_OFFSET+1)
STDFU_BADPARAMETER                  (STDFU_ERROR_OFFSET+2)
STDFU_NOTIMPLEMENTED                (STDFU_ERROR_OFFSET+3)
STDFU_ENUMFINISHED                  (STDFU_ERROR_OFFSET+4)
STDFU_OPENDRIVERERROR               (STDFU_ERROR_OFFSET+5)
STDFU_ERRORDESCRIPTORBUILDING       (STDFU_ERROR_OFFSET+6)
STDFU_PIPECREATIONERROR             (STDFU_ERROR_OFFSET+7)
STDFU_PIPERESETERROR                (STDFU_ERROR_OFFSET+8)
STDFU_PIPEABORTERROR                (STDFU_ERROR_OFFSET+9)
STDFU_STRINGDESCRIPTORERROR         (STDFU_ERROR_OFFSET+0xA)
STDFU_DRIVERISCLOSED                (STDFU_ERROR_OFFSET+0xB)
STDFU_VENDOR_RQ_PB                  (STDFU_ERROR_OFFSET+0xC)
STDFU_ERRORWHILEREADING             (STDFU_ERROR_OFFSET+0xD)
STDFU_ERRORBEFOREREADING            (STDFU_ERROR_OFFSET+0xE)
STDFU_ERRORWHILEWRITING             (STDFU_ERROR_OFFSET+0xF)
```

```
STDFU_ERRORBEFOREWRITING              (STDFU_ERROR_OFFSET+0x10)
STDFU_DEVICERESETERROR                (STDFU_ERROR_OFFSET+0x11)
STDFU_CANTUSEUNPLUGEVENT              (STDFU_ERROR_OFFSET+0x12)
STDFU_INCORRECTBUFFERSIZE             (STDFU_ERROR_OFFSET+0x13)
STDFU_DESCRIPTORNOTFOUND              (STDFU_ERROR_OFFSET+0x14)
STDFU_PIPESARECLOSED                  (STDFU_ERROR_OFFSET+0x15)
STDFU_PIPESAREOPEN                    (STDFU_ERROR_OFFSET+0x16)
STDFU_TIMEOUTWAITINGFORRESET          (STDFU_ERROR_OFFSET+0x17)
```

4.3.2 STDFU_GetDeviceDescriptor

4.3.2.1 Prototype

```
DWORD STDFU_GetDeviceDescriptor(DWORD Num,
                                PUSB_DEVICE_DESCRIPTOR pDesc);
```

4.3.2.2 Description

Gets the DFU-device descriptor

4.3.2.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

pDesc: buffer the descriptor will be copied to.

4.3.2.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code)

4.3.3 STDFU_GetDFUDescriptor

4.3.3.1 Prototype

```
typedef struct _DFU_FUNCTIONAL_DESCRIPTOR
{
  UCHAR bLength;
  UCHAR bDescriptorType;
  UCHAR bmAttributes;
  USHORT wDetachTimeOut;
  USHORT wTransfertSize;
} DFU_FUNCTIONAL_DESCRIPTOR, *PDFU_FUNCTIONAL_DESCRIPTOR;

DWORD STDFU_GetDFUDescriptor(   DWORD Num,
                                PDFU_FUNCTIONAL_DESCRIPTOR pDesc,
                                PUINT pNbOfDFUInterface);
```

4.3.3.2 Description

Gets the DFU descriptor

4.3.3.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

pDesc: buffer the DFU descriptor will be copied to.

pNbOfDFUInterface: pointer to the DFU Interface Number

4.3.3.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code)

4.3.4 STDFU_GetStringDescriptor

4.3.4.1 Prototype

```
DWORD STDFU_GetStringDescriptor(DWORD Num,
                                DWORD Index,
                                LPSTR szString,
                                UINT nStringLength);
```

4.3.4.2 Description

Gets the string descriptor

4.3.4.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

nIndex: desired string descriptor Index. If this index is too high, this function will return an error.

szString: buffer the string descriptor will be copied to

nStringLength: buffer size

4.3.4.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.5 STDFU_GetNbOfConfigurations

4.3.5.1 Prototype

```
DWORD STDFU_GetNbOfConfigurations(    DWORD Num,
                                      PUINT pNbOfConfigs);
```

4.3.5.2 Description

Gets Configurations number

4.3.5.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

pNbOfConfigs: pointer to the number of configurations

4.3.5.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.6 STDFU_GetConfigurationDescriptor

4.3.6.1 Prototype

```
DWORD STDFU_GetConfigurationDescriptor( DWORD Num,
                                        UINT nConfigIdx,
                                        PUSB_CONFIGURATION_DESCRIPTOR pDesc);
```

4.3.6.2 Description

Gets the configuration descriptor

4.3.6.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

nConfigIdx: Number of the selected Configuration

pDesc: buffer the descriptor will be copied to.

4.3.6.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.7 STDFU_GetNbOfInterfaces

4.3.7.1 Prototype

```
DWORD STDFU_GetNbOfInterfaces(  DWORD Num,
                                UINT nConfigIdx,
                                PUINT pNbOfInterfaces);
```

4.3.7.2 Description

Gets the number of interfaces

4.3.7.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

nConfigIdx : Number of the selected Configuration

pNbOfInterfaces: pointer to Number of interfaces

4.3.7.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.8 STDFU_GetNbOfAlternates

4.3.8.1 Prototype

```
DWORD STDFU_GetNbOfAlternates(  DWORD Num,
                                UINT nConfigIdx,
                                UINT nInterfaceIdx,
                                PUINT pNbOfAltSets);
```

4.3.8.2 Description

Gets Alternate Settings Number.

4.3.8.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

nConfigIdx: Number of the selected Configuration

nInterfaceIdx : Number of the selected Interface

pNbOfAltSets: pointer to Alternate Settings Number

4.3.8.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.9 STDFU_GetInterfaceDescriptor

4.3.9.1 Prototype

```
DWORD STDFU_GetInterfaceDescriptor(   DWORD Num,
                                      UINT nConfigIdx,
                                      UINT nInterfaceIdx,
                                      UINT nAltSetIdx,
                                      PUSB_INTERFACE_DESCRIPTOR pDesc);
```

4.3.9.2 Description

Gets the interface descriptor.

4.3.9.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices)

nConfigIdx: Number of the selected Configuration

nInterfaceIdx: Number of the selected Interface

nAltSetIdx: Number of the selected Alternate Setting

pDesc: buffer the descriptor will be copied to.

4.3.9.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.10 STDFU_Open

4.3.10.1 Prototype

```
DWORD STDFU_Open (   DWORD Num,
                     PHANDLE phDevice);
```

4.3.10.2 Description

Opens the DFU driver, giving access to its descriptors.

4.3.10.3 Parameters

Num: Number of the selected DFU Device (Given by STDFU_EnumGetNbDevices).

phDevice: handle returned by the function when the driver is successfully opened

4.3.10.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.11 STDFU_Close

4.3.11.1 Prototype

```
DWORD STDFU_Close(PHANDLE phDevice);
```

4.3.11.2 Description

Closes the DFU driver.

4.3.11.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

4.3.11.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.12 STDFU_Detach

4.3.12.1 Prototype

```
DWORD STDFU_Detach (       PHANDLE phDevice,
                           USHORT wTimeout,
                           UCHAR DFUInterfaceNumber);
```

4.3.12.2 Description

Issues a Detach request to the Control Pipe (Endpoint0).

4.3.12.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

wTimeout: Detach timeout value

DFUInterfaceNumber: DFU Interface Number given by STDFU_GetDFUDescriptor

4.3.12.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.12.5 Remarks

This function sends a detach then it waits for a reset from the device in order to be in a DFU mode, if 15 seconds elapse with no device change, it will return a STDFU_TIMEOUTWAITINGFORRESET error.

4.3.13 STDFU_Dnload

4.3.13.1 Prototype

```
DWORD STDFU_Dnload(       PHANDLE phDevice,
                          UCHAR *pBuffer,
                          ULONG nBytes,
                          USHORT nBlock);
```

4.3.13.2 Description

Issues a Download request to the Control Pipe (Endpoint0).

4.3.13.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

Buffer: Buffer of Data

nBytes: Number of data bytes to be downloaded

nBlock: Number of data blocks to be downloaded

4.3.13.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.14 STDFU_UPload

4.3.14.1 Prototype

```
DWORD STDFU_Upload (      PHANDLE phDevice,
                          UCHAR *pBuffer,
                          ULONG nBytes,
                          USHORT nBlock);
```

4.3.14.2 Description

Issues an Upload request to the Control Pipe (Endpoint0).

4.3.14.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

Buffer: Buffer of Data

nBytes: Number of data bytes to be uploaded

nBlock: Number of data blocks to be uploaded

4.3.14.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.15 STDFU_Getstatus

4.3.15.1 Prototype

```
typedef struct
{
 UCHAR bStatus;
 UCHAR bwPollTimeout[3];
 UCHAR bState;
 UCHAR iString;
}DFUSTATUS, *PDFUSTATUS;

DWORD STDFU_Getstatus (PHANDLE phDevice, DFUSTATUS *DfuStatus);
```

4.3.15.2 Description

Issues a GetStatus request to the Control Pipe (Endpoint0).

4.3.15.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

DfuStatus: structure containing DFU Status structure

4.3.15.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.15.5 Remarks

This function sends a getstatus request and it tests if the device is on a state equal to STATE_DFU_MANIFEST_WAIT_RESET or STATE_DFU_MANIFEST then it closes the driver and it waits for a reset from the device in order to return to application mode, if 15 seconds elapse with no device change, it will return a STDFU_TIMEOUTWAITINGFORRESET error.

4.3.16 STDFU_Clrstatus

4.3.16.1 Prototype

```
DWORD STDFU_Clrstatus (PHANDLE phDevice);
```

4.3.16.2 Description

Issues a ClearStatus request to the Control Pipe (Endpoint0).

4.3.16.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

4.3.16.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.17 STDFU_Getstate

4.3.17.1 Prototype

```
DWORD STDFU_Getstate (PHANDLE phDevice, UCHAR *pState);
```

4.3.17.2 Description

Issues a GetState request to the Control Pipe (Endpoint0).

4.3.17.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function

pState: pointer to a DFU State

4.3.17.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.18 STDFU_Abort

4.3.18.1 Prototype

```
DWORD STDFU_Abort   (PHANDLE phDevice);
```

4.3.18.2 Description

Issues an Abort request to the Control Pipe (Endpoint0).

4.3.18.3 Parameters

phDevice: pointer to handle returned by the STDFU_Open function.

4.3.18.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.19 STDFU_HandleDeviceChange

4.3.19.1 Prototype

```
BOOL STDFU_HandleDeviceChange  (WPARAM wParam,
                                LPARAM lParam);
```

4.3.19.2 Description

This function must be called by the application when receiving a WM_DEVICECHANGE.

4.3.19.3 Parameters

wParam: device-change event that could be for example a DBT_DEVICEARRIVAL or a BT_DEVICEREMOVECOMPLETE event etc.

lParam: event-specific data.

4.3.19.4 Returned value

STDFU_NOERROR in case of success, otherwise (Error Code).

4.3.19.5 Remarks

In any application or GUI working with a driver and thus with a device, you first have to identify and specify the device or the type of the device that you are dealing with and for which a window will receive notifications.

Once this has been done, the WM_DEVICECHANGE device message will notify the application of a change to the hardware configuration of a device.

Note: For more details about this mechanism, please refer to Microsoft MSDN Help, you can also refer to the source files of the DFU demonstrator to see an example of implementation.

5 EXAMPLE VENDOR UPGRADE SOFTWARE (DFU DEMONSTRATOR)

This section describes the DFU demonstrator software, as previously mentioned, this application is divided into two distinct modules: a protocol dynamic link library and a graphical user interface.

5.1 ST7DFUPRT PROGRAMMING INTERFACE

5.1.1 Overview

This DLL is in charge of the protocol layer management between the firmware image (stored on local disk) and data sent/received to/from the ST7 USB device through the ST7DFU.dll generic module by means of DFU requests to the Control Pipe (Endpoint0).

The main features of this DLL are the following :

– Reading & decoding the new firmware image (processing the suffix and useful binary data).

– Writing & coding the old firmware image.

– Performing the firmware upgrade operations by responding to all DFU interface states and the transitions between them.

– Sending to upper layers status, errors, warning messages and progress.

The firmware image will encapsulate the following information : target Addresses, data record sizes, useful data and a suffix.

The purpose of the DFU suffix is to allow the host software to detect and prevent attempts to download incompatible firmware.

For more details about the DFU File suffix and how to compute the CRC please refer to Device Class Specification for DFU Version 1.0 , Appendix B.

Note: Due to the fact that all functions exported by this DLL take a longer time than other modules, its structure will be based on (mainly asynchronous) processes and threaded code.

5.1.2 Programming Interface

This is the application programming interface of the STDFUPRT.dll module. The documentation describes the interface in C language .

5.1.2.1 ST_ImageFromDFUFile

Prototype

```
DWORD WINAPI ST_ImageFromDFUFile(     CHAR * pPathFile,
                                      UCHAR * pBuffer,
                                      UINT * pVid,
                                      UINT * pPid,
                                      UINT * pBcd );
```

Description

Reads and decodes the file image stored in disk and stores all its binary data contents to a buffer in RAM and extracts useful device information (Vendor ID, Product ID, and firmware version)

Parameters

pPathFile : the entry path name of the firmware image stored on disk.

pBuffer : a pointer to a data buffer where the useful data will be stored in RAM.

pVid: a pointer to the Vendor ID parameter.

pPid: a pointer to the productID parameter.

pBcd:a pointer to the firmware version parameter.

Returned Value

STDFU_NOERROR in case of success, otherwise (Error Code).

5.1.2.2 ST_ImageToDFUFile

Prototype

```
DWORD WINAPI ST_ImageToDFUFile( CHAR * pPathFile,
                                UCHAR * pBuffer,
                                UINT nVid,
                                UINT nPid,
                                UINT nBcd );
```

Description

Writes and codes data stored in RAM buffer and received from the device during the up-loading process to a valid firmware image on local disk.

Parameters

pPathFile : the entry path name of the firmware image where the file will be stored on disk.

pBuffer : a pointer to a data buffer received from the device.

nVid : device Vendor ID.

nPid : device product ID.

nBcd: old firmware version.

Returned Value

STDFU_NOERROR in case of success, otherwise (Error Code).

5.1.2.3 ST_DeviceFirmwareUpgrade

Prototype

```
DWORD WINAPI ST_DeviceFirmwareUpgrade(DWORD Nb,
                                      UCHAR * pBufferIn,
                                      UCHAR * pBufferOut);
```

Description

This is the main DFU protocol function, you can use it and/or develop your own solution and it is only intended as an example. Its functional goal is to start all the previous DFU request functions and process the device upgrade automatically.

Parameters

Nb :Number of the selected DFU Device (given by STDFU_EnumGetNbDevices)

pBufferIn : pointer to a 64KByte RAM buffer where the new firmware image is stored.

pBufferOut :pointer to a 64KByte RAM buffer where the oldfirmware image will be stored.

Returned Value

STDFU_NOERROR in case of success, otherwise (Error Code).

5.1.2.4 ST_GetDFUStatus

Prototype

```
DWORD WINAPI ST_GetDFUStatus(          PDWORD pPercent,
                                       PDWORD pStage,
                                       PDWORD pError);
```

Description

Returns DFU status and errors data to the application GUI.

Parameters

pPercent: stage process completion percentage.

pStage:DFU process stage such as detach, upload, upgrade etc.

pError: DFU error.

Returned Value

STDFU_NOERROR in case of success, otherwise (Error Code).

5.2 DFU DEMONSTRATOR GUI

The graphical user interface in this package is provided as a demonstrator and accesses the lower layers of the DFU software(ST7DFUPRT.dll and ST7DFU.Dll) to get information related to the USB device, to initiate DFU operations (Upgrade with or without Upload) and also to display errors, warnings, debug messages and progress .

A simple GUI is provided to demonstrate the DFU utility. Its main features are the following:

– Select one available USB device and display its properties (ex: Vid, Pid, rev),

– Choose to Upgrade the device and to store its present firmware version,

– Select from the PC browser the new firmware image and the file to be stored,

– Integrate a DFU file generator,

– Start button to start operations,

– Abort button to cancel a programming or read operation.

– Progress tool bar,

– Display all debugging messages (Error messages etc.),

– GUI exit, GUI help.

Note: The DFU demonstrator GUI is written in C++, using MFC in Microsoft Visual C++ 6.0. All source code, including a project file is included with the DFU-DK package.

THE PRESENT NOTE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT OF SUCH A NOTE AND/OR THE USE MADE BY CUSTOMERS OF THE INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS."