MCCI Corporation
3520 Krums Corners Road
Ithaca, New York 14850 USA
Phone +1-607-277-1029
Fax +1-607-277-6844
www.mcci.com

DataPump New DFU Protocol User's Guide

Engineering Report 950000298
Rev. C
Date: 2011/09/24

NOTE: The code sections presented in this document are intended to be a facilitator in understanding the technical details. They are for illustration purposes only, the actual source code may differ from the one presented in this document.

Document Release History

Rev. A	2/14/2006	Original release
Rev. B	2011/04/15	Changed all references to Moore Computer Consultants, Inc. to MCCI Corporation. Changed document numbers to nine digit versions. Added missing members of `USBPUMP_PROTOCOL_INIT_NODE structure.` Corrected the macros USTATESWITCH_USBNDFU_DECLARE_..._V1 to USTATESWITCH_NDFU_DECLARE_..._V1 Added missing parameters for the Generic Edge IOCTLs and Edge Download Read IOCTL

Rev. C	2011/09/24	Added source code disclaimer.

TABLE OF CONTENTS

1 Introduction

This Guide presents programming information for developing USB devices that are capable of upgrading the firmware using the MCCI NDFU Protocol Library.

This Guide does not discuss host software, beyond mentioning that MCCI also has drivers and a library for Windows that will work in conjunction with this packet. For more information, please refer to [WINDFU].

1.1 Glossary

USB Universal Serial Bus

USB-IF USB Implementer's Forum, the consortium that owns the USB specification, and which governs the development of device classes.

USBRC MCCI's USB Resource Compiler, a tool that converts a high-level description of a device's descriptors into the data and code needed to realize that device with the MCCI USB DataPump.

1.2 Referenced Documents

[DPIOCTL] *OVERVIEW-ioctl.txt*, from USB DataPump installation `usbkern/doc/` directory.

[DPREF] *MCCI USB DataPump User's Guide*, MCCI Engineering Report 950000066

[DPUSBRC] *USBRC User's Guide*, MCCI Engineering Report 950000061

[USBCORE] *Universal Serial Bus Specificatio*n, version 2.0/3.0 (also referred to as the *USB Specificatio*n). This specification is available on the World Wide Web site **http://www.usb.org**.

[USBDFU] *Universal Serial Bus Device Class Specification for Device Firmware Upgrade,* version 1.1 (also referred to as the *DFU Specificatio*n, where *"DFU"* stands for "Device Firmware Upgrade". This specification is available at **http://www.usb.org/developers/devclass.**

[WINDFU] *DFU Library for Windows User's Guide*, MCCI Engineering Report 950000196

1.3 Overview

The MCCI NDFU Protocol Library, in conjunction with the MCCI USB DataPump, provides a straightforward, portable environment for implementing firmware (or other non-volatile data) download and/or upload over USB using the USB DFU 1.1 protocol. The NDFU Protocol can be used to create a stand-alone device, or can be combined with other MCCI and user-provided protocol code to create multi-function devices.

This document describes the portions of the MCCI NDFU Protocol Library that are visible to an external client. As such, it serves as a Library User's Guide. It is not intended to serve as a stand-alone reference, but it should be used in conjunction with the MCCI DataPump User's Guide [DPREF] and the USB DFU 1.1 Specification [USBDFU]. The purpose of the NDFU Library is to encapsulate issues regarding USB transactions so that the user can concentrate on the DFU portions of a target device.

1.4 Differences Between DFU and NDFU

The differences between MCCI DFU Protocol Library and MCCI NDFU Protocol Library are

- NDFU Library API defined in terms of abstract DOWNLOAD class.
- Different calling mechanism in Protocol API
- NDFU allows the use of a bulk-pipe for data transport. This is a functional extension from [USBDFU].

1.5 Initialization and Setup

When using the DataPump NDFU Protocol, the final application consists of two distinct parts. The first part is provided by MCCI and consists of the MCCI USB DataPump libraries and specifically, the MCCI USB NDFU Protocol Library. This document uses the name **Protocol** to refer collectively to these components. The second part is provided by the developer and consists of application and device specific modules. This document uses the name **Client** to refer to these components.

1.5.1 Protocol Library Initialization

The Protocol Library code parses the device descriptors, and creates Protocol instances for each supported DFU function. The Protocol NDFU Class functions are represented by an interface descriptor with bInterfaceClass 0xFE, bSubClass 0x01 and bInterfaceProtocol 0x00. These codes indicate to the library:

- that the interface represents an Application Specific Class device (bInterfaceClass 0xFE),
- that the interface class is DFU (bSubClass 0x01), and
- that no specific protocol runs on the interface (pInterfaceProtocol 0x00).

Each such interface may also supply two bulk endpoints, an IN endpoint and an OUT endpoint, to be used by the Protocol Library for data transport. NDFU is not sensitive to the order of the endpoints in the descriptor set, nor to the wMaxPacketSize of the endpoints.

Note that there should be no more than one DFU Class Interface within each configuration (see [USBDFU] section 4.1).

The following fragment of USBRC code shows how this might be coded:

```
interface 0
        {
        class    0xFE           #Application Specfic Class
        subclass 0x01           #DFU
        protocol 0x00           #No specific protocol
        name     S_DFUDEV1      #string reference
        ;
        }
```

or, if optional bulk-pipe shall be used for data-transport,

```
interface 0
        {
        class    0xFE           #Application Specfic Class
        subclass 0x01           #DFU
        protocol 0x00           #No specific protocol
        name     S_DFUDEV1      #string reference

        endpoints
                bulk in  1 packet-size 64
                bulk out 1 packet-size 64
                ;
        }
```

The Protocol Library will create one Protocol instance for each supported DFU interface that it finds in the descriptor set. If a DFU class interface appears in multiple configurations, then the Protocol Library will create multiple instances, one for each configuration. Each Protocol instance will be named according to the abstract DOWNLOAD protocol class and added to USB DataPump object dictionary.

The Protocol instance code performs all command set decoding, however it contains no code that actually knows how to read and write data blocks. For this purpose, the system integrator must provide Client code. This is discussed in the next section.

Finally, the USB DataPump must be instructed to include NDFU support in the code being built. This is done using the application initialization vector. See section 2.1 below.

1.5.2 Client Instance Initialization

When the DataPump has been initialized, the target operating system must discover the available DOWNLOAD protocol instances using the USB DataPump object dictionary, and create Client instances. Each Client instance registers with a Protocol instance. All communication from Client to Protocol is accomplished using a downcall I/O-control mechanism, known as an **IOCTL**, defined by the DataPump and implemented by the Protocol (see section 4). When a function in the Client needs to access a service in the Protocol, then a call is made to a Protocol Library function to send down the appropriate service code using the IOCTL mechanism.

Because USB device firmware is controlled by the host PC, there is a need for asynchronous communication from the Protocol instance to the Client instance. Communications from Protocol to Client are accomplished using an upcall I/O-control mechanism, known as an **Edge-IOCTL**. The IOCTLs are defined by the DataPump and are routed by the DataPump to a function supplied by the Client during the initialization process (see section 2.3). When a function in the Protocol needs to access a service in the Client a call is made to the Edge-IOCTL mechanism supplied with the appropriate service code.

During initialization, the Client will receive control from the platform startup code. The Client is then responsible for enumerating and initializing all instances of the Protocol by repeatedly calling

```
UsbPumpObject_EnumerateMatchingNames(
        ...,
        USBPUMP_OBJECT_NAME_ENUM_DOWNLOAD,
        ...)
```

Each time the function returns a non-NULL pointer to a Protocol `USBPUMP_OBJECT_HEADER`, the Client code must

- Create a matching Client instance, with an accompanying `USBPUMP_OBJECT_HEADER` to represent the Client instance to the DataPump.
- Call `UsbPumpObject_Init()` to initialize the Client instance `USBPUMP_OBJECT_HEADER` and bind it to the Edge-IOCTL function provided by the Client.
- Call `UsbPumpObject_FunctionOpen()` to open the Protocol object and bind it to the Client instance object. The `USBPUMP_OBJECT_HEADER` pointer returned by the call is the reference that the Client instance will use to access the Protocol instance thru the IOCTL mechanism.

Please also refer to DataPump Professional and Standard source installation usbkern/doc/OVERVIEW-appinit.txt and OVERVIEW-objects.txt.

Applications wishing to make use of the Protocol library should

- include the header file `usbdfu10.h`, `ufnapidownload.h` and `usbioctl_download.h`
- link with library `protondfu`.

2 Data structures

Several data structures are involved in initializing and running the Protocol. The ones that are of interest for the Client are listed below.

2.1 USBPUMP_PROTOCOL_INIT_NODE

This structure is part of the `USB_DATAPUMP_APPLICATION_INIT_VECTOR_HDR` that the Client passes to the DataPump init function. It is preferably initialized using `USBPUMP_PROTOCOL_INIT_NODE_INIT_V2` since this provides backward compatibility with future releases of the DataPump.

This structure is used by the enumerator to match the Protocol against the device, configuration and interface descriptors when locating interfaces to use for the Protocol, and to bind init functions to the Protocol. The fields of interest to the Client are:

`sDeviceClass:`	Normally –1 → allows matching to any device class.
`sDeviceSubClass:`	Normally –1 → allows matching to any device subclass
`sDeviceProtocol:`	Normally –1 → allows matching to any device protocol
`sInterfaceClass:`	`USB_bInterfaceClass_Dfu`
`sInterfaceSubClass:`	`USB_bInterfaceSubClass_Dfu`
`sInterfaceProtocol:`	Normally –1 → allows matching no matter what bInterfaceProtocol is used
`sConfigurationValue:`	Normally –1 → allows matching no matter what bConfigurationValue was used in the configuration descriptor
`sInterfaceNumber:`	Normally –1 → allows matching no matter what bInterfaceNumber is on the interface.
`sAlternateSetting:`	Normally –1 → allows matching no matter what bAlternateSetting is on the interface
`sSpeed:`	Always –1 (Reserved for future use)
`uProbeFlags`	Flags that control the probing of multiple instances.
`pProbeFunction:`	Optional pointer to `USBPUMP_PROTOCOL_PROBE_FN` function. If this function is available and returns FALSE then the `pCreateFunction` function will not be called prohibiting the creation of the protocol instance.
`pCreateFunction:`	Normally `NDfu_ProtocolCreate` – this function will create the appropriate set of protocol objects to implement the appropriate class-level behavior.
`pQualifyAddInterfaceFunction`	Optional add-instance qualifier function. If this function is available and returns TRUE then `pAddInterfaceFunction` will be called to add the interface

`pAddInterfaceFunction`	Optional function for adding instance
`pOptionalInfo:`	Pointer to UPROTO_NDFU_CONFIG structure (see section 2.2)

2.2 UPROTO_NDFU_CONFIG

This structure is pointed to by the `USBPUMP_PROTOCOL_INIT_NODE`. It is preferably initialized using the macro `UPROTO_NDFU_CONFIG_INIT_V1` since this provides backward compatibility with future releases of the Protocol.

This structure is used to configure the Protocol. The fields of interest to the Client are:

`pStateSwitch:`	Pointer to `USTATESWITCH_NDFU` containing NDFU statehandler functions. See section 2.3
`pDeviceMode:`	Pointer to USBPUMP_DOWNLOAD_DEVICE_MODE indicating which DFU mode the device is running in

2.3 USTATESWITCH_NDFU

This structure is pointed to by the `UPROTO_NDFU_CONFIG`. It is preferably initialized using one of the `USTATESWITCH_NDFU_DECLARE_..._V1` macros in the header file `ndfucfg.h` since this provides backward compatibility with future releases of the Protocol.

A properly selected state switch, that matches the Protocols device mode (Application- or DFU-Mode) and functionality configured in the DFU functional descriptor, reduces program memory requirements that can be important in embedded systems.

The macros in header file `ndfucfg.h` that can help a Client create this switch are:

`USTATESWITCH_NDFU_DECLARE_ALLSTATES_V1( Prefix)`	Declares and initializes a StateSwitch structure containing all states (both Application and DFU mode). The structure name will be “Prefix”_kAllStateSwitch.
`USTATESWITCH_NDFU_DECLARE_APPMODE_V1( Prefix)`	Declares and initializes a StateSwitch structure containing only Application-mode states. The structure name will be “Prefix”_kAppModeStateSwitch.
`USTATESWITCH_NDFU_DECLARE_DFUMODE_DNLD_V1( Prefix)`	Declares and initializes a StateSwitch structure containing only DFU-mode states used to download data. The structure name will be “Prefix”_kDfuModeDnLdStateSwitch.
`USTATESWITCH_NDFU_DECLARE_DFUMODE_UPLD_V1( Prefix)`	Declares and initializes a StateSwitch structure containing only DFU-mode states used to upload

	data. The structure name will be "Prefix"_kDfuModeUpLdStateSwitch.
`USTATESWITCH_NDFU_DECLARE_DFUMODE_DNUPLD_V1( Prefix)`	Declares and initializes a StateSwitch structure containing all DFU-mode states (both download and upload). The structure name will be "Prefix"_kDfuModeDnUpLdStateSwitch.

3 Edge IOCTL (Upcall) services

The following section describes the services the Client must provide to the Protocol thru the Edge-IOCTL function given when initializing the Client object using `UsbPumpObject_Init()` (see section 1.5.2).

3.1 Edge IOCTL function

```
Type name :     USBPUMP_OBJECT_IOCTL_FN

Prototype :     USBPUMP_IOCTL_RESULT My_IOCTL(
                    USBPUMP_OBJECT_HEADER *p,    /* Pointer to target obj */
                    USBPUMP_IOCTL_CODE,          /* IOCTL-code */
                    CONST VOID *,                /* Pointer to in parameter */
                    VOID *                       /* Pointer to out parameter */
                    );

Header-file : usbpumpobject.h
```

3.2 Generic Edge IOCTLs

3.2.1 Edge Activate

IOCTL code	USBPUMP_IOCTL_EDGE_ACTIVATE
In parameter structure	CONST USBPUMP_IOCTL_EDGE_ACTIVATE_ARG *
Field pObject	Pointer to lower-level UPROTO object header
Field pClientContext	Context handle supplied by client when it is connected to the lower-level UPROTO object
Out parameter	USBPUMP_IOCTL_EDGE_ACTIVATE_ARG *
Field fReject	If set TRUE, then the Client would like the Protocol to reject the request, if possible. Note that fReject is an advisory indication, which may be used to flag to the Protocol that the Client cannot actually operate the data streams at this time.

	Because of hardware or protocol limitations, this might or might not be honored by the lower layers. Field is initialized to FALSE by Protocol.
Description	This IOCTL is sent from Protocol to Client whenever the host does something that brings up the Protocol instance.
Note	The out parameter is initialized by the Protocol with the same values as the in parameter

3.2.2 Edge Deactivate

IOCTL code	USBPUMP_IOCTL_EDGE_DEACTIVATE
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DEACTIVATE_ARG *
Field pObject	Pointer to lower-level UPROTO object header
Field pClientContext	Context handle supplied by client when it is connected to the lower-level UPROTO object
Out parameter	NULL
Description	The Protocol issues this IOCTL whenever a (protocol-specific) event occurs that deactivates the Protocol instance. Unlike the ACTIVATE call; the Client has no way to attempt to reject this call. The USB host might have issued a reset -- there's no way to prevent, in general, deactivation.

3.2.3 Edge Bus Event

IOCTL code	USBPUMP_IOCTL_EDGE_BUS_EVENT
In parameter structure	CONST USBPUMP_IOCTL_EDGE_BUS_EVENT_ARG *
Field pObject	Pointer to lower-level UPROTO object header
Field pClientContext	Context handle supplied by client when it is connected to the lower-level UPROTO object
Field EventCode	Instance of UEVENT. The type of event that occurred. This will be one of UEVENT_SUSPEND, UEVENT_RESUME, UEVENT_ATTACH, UEVENT_DETACH, or UEVENT_RESET. [UEVENT_RESET is actually redundant; it will also cause a deactivate event; however this hook may be useful for apps that wish to model the USB state.]
Field pEventSpecificInfo	The event-specific information accompanying the UEVENT.Pointer to Client

	specific event info. See “ueventnode.h” for details.
Field fRemoteWakeupEnable	Set TRUE if remote-wakeup is enabled.
Out parameter	NULL
Description	Whenever a significant bus event occurs, the Protocol will arrange for this IOCTL to be made to the Client. Any events that actually change the state of the Protocol will also cause the appropriate Edge-IOCTL to be performed; SUSPEND and RESUME don't actually change the state of the Protocol (according to the USB core spec).

3.3 Download specific Edge IOCTLs

Common IN parameter fields for all Edge Download IOCTLs are

Field pObject	Pointer to Client object
Field pClientContext	Pointer to Client context

Common OUT parameter fields for all Edge Download IOCTLs are

Field Status	Return status from Client
Field fReject	Set TRUE to reject request. Field initialized to FALSE by Protocol
Note	The out parameter is initialized by the Protocol with the same values as the in parameter

3.3.1 Edge Download Start Tmr

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_START_TMR
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_START_TMR_ARG *
TmoInMs	Timeout in milliseconds
Id	Timer id - Value returned on timeout
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_START_TMR_ARG *
Description	This IOCTL is sent from Protocol to Client whenever the Protocol needs to start a timer. Current version of the Protocol uses this IOCTL while in DFU-state 1 appDETACH (see [USBDFU] section 5.1). Client is expected to call UsbFnApiDownload_Timeout() (see section 4.3) after “TmoInMs” milliseconds has elapsed. The Protocol may send this Edge-IOCTL again before the timeout in “TmoInMs” has elapsed. The Client should in such

case restart the timer with the new timeout value and update the timer id.

3.3.2 Edge Download Open Write

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_WRITE
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_WRITE_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_WRITE_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_2_DFU_WRITE_IDLE` indicates that the Client is ready to receive data. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the `STATUS IOCTL`. The Protocol will send a `RESET IOCTL` to reinitialize the Client to its initial state effectively terminating the write operation.
Field NextTmoInMs	Number of milliseconds the Client will need to complete the first `WRITE` Edge-`IOCTL`
Description	This IOCTL is sent from Protocol to Client as part of the of the transition between DFU-state 2 dfuIDLE and state 3 dfuDNLOAD-SYNC. The Client is expected to make appropriate actions in order to be prepared for receiving data to the device.

3.3.3 Edge Download Write

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_WRITE
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_WRITE_ARG *
Field nBytes	Number of bytes to write to the device
Field pBuf	Pointer to buffer
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_WRITE_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_2_DFU_WRITE_IDLE` indicates that the Client has completed the write operation and is ready to receive more data. `USBPUMP_DOWNLOAD_CLIENT_STATE_3_DFU_WRITE_BUSY` indicates that the Client has not completed the write operation and is not ready to receive more data. The Protocol will send `WRITE` Edge-`IOCTL` again (“pBuf” set to `NULL` and “nBytes” set to zero). This way the Client may dynamically

	acquire as much time as it needs for erasing and programming a region of memory. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the `STATUS` Edge-`IOCTL`. The Protocol will send `RESET` Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the write operation.
Field NextTmoInMs	Number of milliseconds the Client will need to complete the next `WRITE` Edge-`IOCTL`
Description	This IOCTL is sent from Protocol to Client as long as the host has data to download. The Client must return from the `IOCTL` within the timeout indicated in the previous Edge-`IOCTL` (`OPEN_WRITE` or `WRITE`)

3.3.4 Edge Download Close Write

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_WRITE
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_WRITE_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_WRITE_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_4_DFU_MANIFEST_BUSY` indicates that the Client is ready to start the manifestation phase. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the `STATUS` Edge-`IOCTL`. The Protocol will send `RESET` Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the write operation.
Field NextTmoInMs	Number of milliseconds the Client will need to complete the first `MANIFEST` Edge-`IOCTL`
Description	This IOCTL is sent from Protocol to Client when the host has no more data to download. The Client is expected make appropriate actions in order to be prepared for the manifestation phase of the download (if manifestation is required by device), and then immediately return from the `IOCTL`.

3.3.5 Edge Download Open Read

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_READ
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_READ_ARG *

Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_OPEN_READ_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_5_DFU_READ_BUSY` indicates that the Client is ready to transmit data. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the STATUS Edge-`IOCTL`. The Protocol will send RESET Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the read operation.
Description	This IOCTL is sent from Protocol to Client as part of the of the transition between DFU-state 2 dfuIDLE and state 9 dfuUPLOAD-IDLE. The Client is expected to make appropriate actions in order to be prepared for transmitting data from the device.

3.3.6 Edge Download Read

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_READ
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_READ_ARG *
Field nMaxBytes	The maximum number bytes to read from the device
Field pBuf	Pointer to buffer
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_READ_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_5_DFU_READ_BUSY` indicates that the last read operation was successful. The Client can transmit more data as long as “nBytes” = “nMaxBytes”. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the STATUS Edge-`IOCTL`. The Protocol will send RESET Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the read operation.
Field nBytes	The actual number bytes read from the device
Description	This IOCTL is sent from Protocol to Client as long as the Client returns “nBytes” = “nMaxBytes”. The Client must promptly return from the `IOCTL`.

3.3.7 Edge Download Close Read

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_READ

In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_READ_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_CLOSE_READ_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_1_DFU_IDLE` indicates that the Client is ready and back to idle again. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the `STATUS` Edge-`IOCTL`. The Protocol will send `RESET` Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the read operation.
Description	This IOCTL is sent from Protocol to Client when the Client has returned less than requested amount of data for upload. The Client is expected to make appropriate actions in order to clean up the read operation, and then immediately return from the `IOCTL`.

3.3.8 Edge Download Manifest

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_MANIFEST
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_MANIFEST_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_MANIFEST_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_1_DFU_IDLE` indicates that the Client has finished the manifestation phase. `USBPUMP_DOWNLOAD_CLIENT_STATE_4_DFU_MANIFEST_BUSY` indicates that the Client is not ready with the manifestation phase. `USBPUMP_DOWNLOAD_CLIENT_STATE_6_DFU_ERROR` indicates that something is wrong, and that the Protocol may find out the cause by sending the `STATUS` Edge-`IOCTL`. The Protocol will send `RESET` Edge-`IOCTL` to reinitialize the Client to its initial state effectively terminating the manifest operation.
Field NextTmoInMs	Number of milliseconds the Client will need to complete the next `EDGE_MANIFEST IOCTL`
Description	This IOCTL is sent from Protocol to Client after a successful call to `CLOSE_WRITE` Edge-`IOCTL` or as long as the Client indicates that it isn't ready with the manifestation phase. Some devices that cannot program memory while maintaining the USB function may store data in some intermediate storage while in the write phase, and then use the manifestation phase to transfer data into nonvolatile memory. Other devices that are capable of programming memory while maintaining the USB function can use this phase to verify that the downloaded data is valid, i.e.

	has a correct checksum.

3.3.9 Edge Download Status

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_STATUS
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_STATUS_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_STATUS_ARG *
Field ClientState	Client state on return from `IOCTL`.
Field ClientStatus	Client status on return from `IOCTL`.
Field iString	If ClientStatus is `USB_Dfu_Status_errVENDOR` then iString must be set to an index of a string descriptor with vendor specific information on the Client status.
Description	This IOCTL is sent from Protocol when it needs to find out the Client's internal state and status

3.3.10 Edge Download Reset

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_RESET
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_RESET_ARG *
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_RESET_ARG *
Field ClientState	Client state on return from `IOCTL`. `USBPUMP_DOWNLOAD_CLIENT_STATE_1_DFU_IDLE` indicates that the Client has reset itself and is ready for a new operation.
Description	This IOCTL is sent from Protocol when it needs to reset the Client to a known state.

3.3.11 Edge Download Restart

IOCTL code	USBPUMP_IOCTL_EDGE_DOWNLOAD_RESTART
In parameter structure	CONST USBPUMP_IOCTL_EDGE_DOWNLOAD_RESTART_ARG *
Field DeviceMode	`USBPUMP_DOWNLOAD_DEVICE_MODE_APP` indicates that the DFU interface should try to start up in Application-Program-Mode the next time (see [USBDFU] Figure A.1).

	`USBPUMP_DOWNLOAD_DEVICE_MODE_DFU` indicates that the DFU interface should start up in DFU-Program-Mode the next time.
Out parameter	USBPUMP_IOCTL_EDGE_DOWNLOAD_RESTART_ARG *
Description	This IOCTL is sent from Protocol when it needs to do a warm-boot for the entire device. The Client is not expected to return from this `IOCTL`, even if doing so is not an error (as long as the Client at least has initiated some kind of warm-boot process that eventually will restart the device). NOTE: The Client may also decide to go to DFU-mode regardless of the DeviceMode if, during startup-test, it finds that the firmware or other data is corrupt and needs to be downloaded again. The Protocol will read the pDeviceMode field of the configuration (see section 2.2) during initialization to find out which mode (Application or DFU) the Client is in. The Client must therefore initialize this pointer correctly, and it is also preferred if it can maintain pre-boot error status information so that the host can find out what went wrong during last operation even if the device had to be warm-booted.

4 Downcall services

The following section describes the services the Protocol provides to the Client thruugh library functions provided by the Protocol.

4.1 Download Calc CRC

```
Prototype :

USBPUMP_IOCTL_RESULT UsbFnApiDownload_CalcCrc(
        USBPUMP_OBJECT_HEADER *                      pIoObject,
        CONST UCHAR *                                pBuf,
        BYTES                                        nBytes,
        UINT32 *                                     pCrcAcc
        );

Header-file : ufnapidownload.h
```

This function is used by Client to calculate a CRC-32 on a piece of memory by calling this function with ”pBuf”, a pointer to the beginning of some memory area, “nBytes” indicating the length in bytes of the memory area, and finally “CrcAcc” which is the initial value used by the CRC (Cyclic Redundancy Check) algorithm. “CrcAcc” must initially be set to `USBPUMP_DOWNLOAD_CRC_START`, and should be set to the return-value of a previous call if the function is being used to calculate the CRC of some non-consecutive memory areas. The final return value may be stored in some non-volatile memory area and used at a later point to verify that the data hasn’t been modified.

The parameters are:

`pIoObject`	This is a pointer to Protocol instance object.
`pBuf`	Pointer to buffer
`nBytes`	Number of bytes in buffer
`pCrcAcc`	Pointer to start/end value of CRC-accumulator

4.2 Download Get Info

```
Prototype :

USBPUMP_IOCTL_RESULT UsbFnApiDownload_GetInfo(
        USBPUMP_OBJECT_HEADER *                        pIoObject,
        UINT8 *                                        pStringId
        );

Header-file : ufnapidownload.h
```

This function is used by Client to get the string descriptor id associated with the interface. It can be used to determine which memory segment this Client instance should act upon. See [USBDFU] section 4.2.3 "DFU Mode Interface Descriptor" comment on alternate settings. The parameters are:

`pIoObject`	This is a pointer to Protocol instance object.
`pStringId`	Pointer to string descriptor id associated with interface

4.3 Download Timeout

```
Prototype :

USBPUMP_IOCTL_RESULT UsbFnApiDownload_Timeout(
        USBPUMP_OBJECT_HEADER *                        pIoObject,
        UINT32                                         Id
        );

Header-file : ufnapidownload.h
```

This function is used by the Client when the timer started by the Edge-IOCTL `USBPUMP_IOCTL_EDGE_DOWNLOAD_START_TMR` has reached its timeout value. The parameters are:

`pIoObject`	This is a pointer to Protocol instance object.
`Id`	Timer id given when started

5 Demo applications

The DataPump Professional and Standard installations contain a demo in `usbkern/app/ndfudemo` and `usbkern/proto/ndfu/applib` that can be used as reference on how to use the NDFU protocol.