Application Note

AN_406

MCCI-USB DataPump Virtual Ethernet Protocol User Guide

Version 1.0

Issue Date: 2017-09-13

This user guide describes the portions of the MCCI Virtual Ethernet Protocol Library that are visible to an external client. As such, it serves as a Library User's Guide.

Bridgetek Pte Ltd (BRTChip)
178 Paya Lebar Road, #07-03, Singapore 409030
Tel: +65 6547 4827 Fax: +65 6841 6071
Web Site: http://www.brtchip.com

MCCI Legal and Copyright Information

Copyright:

Trademark Information:

The following are registered trademarks of MCCI Corporation:

- MCCI
- TrueCard
- TrueTask
- MCCI USB DataPump
- MCCI Catena

The following are trademarks of MCCI Corporation:

- MCCI Skimmer
- MCCI Wombat
- InstallRight
- MCCI ExpressDisk

All other trademarks, brands and names are the property of their respective owners.

Disclaimer of Warranty:

MCCI Corporation ("MCCI") provides this material as a service to its customers. The material may be used for informational purposes only.

MCCI assumes no responsibility for errors or omissions in the information contained at the world wide web site located at URL address 'http://www.mcci.com/', links reachable from this site, or other information stored on the servers 'www.mcci.com', 'forums.mcci.com', or 'news.mcci.com' (collectively referred to as "Web Site"). MCCI further does not warrant the accuracy or completeness of the information published in the Web Site. MCCI shall not be liable for any special, indirect, incidental, or consequential damages, including without limitation, lost revenues or lost profits, which may result from the use of these materials. MCCI may make changes to this Web Site, or to the products described therein, at any time without notice. MCCI makes no commitment to maintain or update the information at the Web Site.

THESE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

Table of Contents

1 Introduction

The MCCI USB DataPump® product is a portable firmware framework for developing USB-enabled devices. As part of the DataPump, MCCI provides a portable, generic implementation of an Ethernet NIC interface that uses the USB Device Working Group CDC Ethernet or Microsoft Remote NDIS protocols. We present the programming information for integrating this support into user's firmware, to create a USB device that presents an Ethernet interface to the host PC.

The host software issues are not discussed here. Since, the MCCI implementation complies with the CDC Ethernet and Microsoft RNDIS standard, most operating system host drivers will work directly with MCCI's implementation. For information on Microsoft Windows support for CDC Ethernet, please refer to Microsoft USB CDC Ethernet FAQ [WINUSBFAQ].

1.1 Overview

The MCCI Virtual Ethernet Protocol Library in conjunction with MCCI CDC Ethernet and/or MCCI RNDIS Protocol Libraries, and the MCCI USB DataPump, provides a straightforward, portable environment for implementing Ethernet devices over USB using the USB CDC Ethernet or Microsoft RNDIS protocol. The MCCI Virtual Ethernet Protocol Library can be used to create a stand-alone device, or can be combined with other MCCI- and/or user-provided protocols to create multi-function devices.

This document describes the portions of the MCCI Virtual Ethernet 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 should be used in conjunction with the AN_402_MCCI_USB_DataPump_UserGuide and Microsoft RNDIS Specifications. The purpose of the Virtual Ethernet Protocol Library is to encapsulate the issues regarding USB transactions so that the user can concentrate on the Ethernet portions of a target device.

1.2 Initialization and Setup

When using the DataPump Virtual Ethernet Protocol, the final application consists of two distinct parts. The first part is provided by MCCI and consists of the MCCI USB DataPump libraries, MCCI CDC Ethernet and/or MCCI RNDIS Protocol Libraries and specifically, the MCCI USB Virtual Ethernet. 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.

1.2.1 Protocol Library Initialization

The Protocol Library code parses the device descriptors, and creates Protocol Instances for each supported Ethernet Class function. The Protocol CDC Ethernet Class functions are represented by an interface descriptor with bInterfaceClass 0x02, bInterfaceProtocol 0x00, and bSubClass 0x06. These codes indicate to the library:

- that the interface represents a CDC Class device (bInterfaceClass 0x02),
- that the command set for the interface is Ethernet (bInterfaceProtocol 0x00), and

Each such interface must also supply 2 bulk and 1 interrupt EP. The following fragment of USBRC code shows how this might be coded:

```
interface 0
      {
      class  0x02             # Communication class
      subclass 0x02                 # Abstract Control
      protocol 0xFF                 # vendor-specific
      name   S_RNDISCOMMIFC         # string

      private-descriptors
            {
            # CDC HEADER functional descriptor
            raw   {
                  0x24        # interface
                  0           # functional descriptor
                  word(0x120) # CDC
                              # version 1.2
                  };

            raw   {
                  0x24  #CS_INTERFACE
                  0x01  #call management.
                  0x00  # no call management internal
                  0x01  # interface # of DataClass interface
                  };

            raw   {
                  0x24  #CS_INTERFACE
                  0x02  #Abstract Control Management descr.
                  0x00  # no capabilities
                  };

            # CDC UNION functional descriptor
            raw   {
                  0x24  #CS_INTERFACE
                  0x06  #union functional descriptor
                  0x00  # interface# for comm class interface
                  0x01  # interface# for data class interface
                  };
            }

      endpoints
            # no need for double-buf, put
            # this one last.
            interrupt in 7 packet-size 64
                  polling-interval 1
            ;
      }

#
# Interface 1 is the (only) data interface; it is
# used for transmitting data frames.
#
interface 1
      {
      class 0x0A        #data class
      subclass 0x00           #none
      protocol 0x00           #none
      name S_RNDISDATA  # string
      endpoints
```

```
                bulk out
                bulk in
                ;
            }
};
```

The protocol library will create one Protocol Instance for each supported Virtual Ethernet interface that it finds in the descriptor set. If a Virtual Ethernet interface appears in multiple configurations, then the protocol library will create multiple instances, one for each configuration.

The Protocol Instance code performs all command set decoding, however it contains no code that actually knows how to read and write data blocks. It also requires assistance for obtaining this from MSC. 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 Virtual Ethernet Protocol support in the code being built. This is done using the application initialization vector. See Section 2.1.

1.2.2 Client Instance Initialization

Client's code dynamically locates Protocol instances using the USB DataPump object dictionary. When the DataPump is initialized, the modules will create protocol instances, and will give those names.

After the DataPump has been initialized, the target operating system must discover the available Virtual Ethernet instances, and must create client instances. Each client instance registers with a protocol instance. All communications from Client to Protocol are 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 the IOCTL mechanism supplied with the appropriate service code.

Since 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 IO-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, then 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_VETHER,
          ...)
```

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.

Applications wishing to make use of the Protocol library should -

- include the header file ufnapivether.h and usbioctl_vether.h
- link with library protovether

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. The macro USBPUMP_PROTOCOL_INIT_NODE_INIT_V2 is preferably used to initialize the structure 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_MassStorage??
sInterfaceSubClass:	USB_bInterfaceSubClass_MassStorageATAPI??
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 CdcSubClass_Ethernet_ProtocolCreate – this function will create the appropriate set of protocol objects to implement the appropriate class-level behavior.
pQualifyAddInterfaceFunction	Pointer to USPBUMP_PROTOCOL_QUALIFY_ADD_INTERFACE_FN function. Optional add-instance qualifier function. If this function is available and return TRUE then pAddInterfaceFunction will be called to add the interface.
pAddInterfaceFunction	Pointer to USPBUMP_PROTOCOL_ADD_INTERFACE_FN function. Optional function for adding instance.
pOptionalInfo:	Pointer to UPROTO_CDCSUBCLASS_xxx_CONFIG structure (see Section 2.2)

2.2 UPROTO_MSCSUBCLASS_ATAPI_CONFIG?? MSC

This structure is pointed to by the USBPUMP_PROTOCOL_INIT_NODE. The macro USBPUMP_PROTOCOL_INIT_NODE_INIT_V2 is preferably used to initialize the structure 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:

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_MassStorage
sInterfaceSubClass:	USB_bInterfaceSubClass_MassStorageATAPI
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 CdcSubClass_Ethernet_ProtocolCreate – this function will create the appropriate set of protocol objects to implement the appropriate class-level behavior.
pQualifyAddInterfaceFunction	Pointer to USPBUMP_PROTOCOL_QUALIFY_ADD_INTERFACE_FN function. Optional add-instance qualifier function. If this function is available and return TRUE then pAddInterfaceFunction will be called to add the interface.
pAddInterfaceFunction	Pointer to USPBUMP_PROTOCOL_ADD_INTERFACE_FN function. Optional function for adding instance.
pOptionalInfo:	Pointer to UPROTO_CDCSUBCLASS_xxx_CONFIG structure

```
https://support.mcci.com/customer/flax/fennec/files/firmware/V3_16a-20160204a/
\usbkern\arch\ft32\os\none\soc\ft900\app\ft900dci_rndiseth\ft900dci_rndiseth_tables.c
CONST USBPUMP_PROTOCOL_INIT_NODE InitNodes[] =    \
 {        \
 USBPUMP_PROTOCOL_INIT_NODE_INIT_V2(    \
  / dev class, subclass, proto / -1, -1, -1,  \
  / ifc class / USB_bInterfaceClass_Comm,  \
  / subclass /  USB_bInterfaceSubClass_CommACM,  \
  / proto / 0xFF,     \
  / cfg, ifc, altset / -1, -1, -1,   \
  / speed / -1,      \
  / probe flags / USBPUMP_PROTOCOL_INIT_FLAG_AUTO_ADD, \
  / probe / UsbPumpProtoAbstractNicCdcRndis_ProtocolProbe,\
  / create / UsbPumpProtoAbstractNicCdcRndis_ProtocolCreate,\
  / qualifyAddInterface / NULL,    \
  / addInterface / UsbPumpProtoAbstractNic_AddDataInterface, \
  / optional info / (VOID *) &AbstractNicCdcRndisConfig \
  )       \
 };
```

2.3 USB_DATAPUMP_APPLICATION_INIT_VECTOR

This structure is pointed to by the USB_DATAPUMP_APPLICATION_INIT_VECTOR_HDR. The macro USB_DATAPUMP_APPLICATION_INIT_VECTOR_INIT_V1 is preferably used to initialize using the structure since this provides backward compatibility with future releases of the Protocol.

UsbPortIndex:	The port index is used for matching up an application with a port. You might not have a symmetrical application -- each USB port might have a different function. Therefore, we allow you to replicate entries for each USB port. An entry of -1 is a wildcard.

pDescriptorTable:	Pointer to this applications descriptor table. The name of the descriptor table is given by the USB Resource file being used
pDeviceInitFunction:	Pointer to this applications descriptor table init function. The name of the init function is given by the USB Resource file being used
sizeof_Udevice:	Size of the device structure for this application. The name of the device structure is given by the USB Resource file being used
DebugFlags	The recommended debug flags
pAppProbeFunction:	The application probe function, if present, is called prior to initializing the device, so that it can decide whether or not to init.
pAppInitFunction:	Usually UsbPump_GenericApplicationInit_Protocols
pOptionalAppInfo:	Pointer to optional USBPUMP_PROTOCOL_INIT_NODE_VECTOR

2.4 USB_DATAPUMP_APPLICATION_INIT_VECTOR_HDR

This structure is used as input to the DataPump OS-specific init function. It is preferably initialized using the macro USB_DATAPUMP_APPLICATION_INIT_VECTOR_HDR_INIT_V1 since this provides backward compatibility with future releases of the Protocol.

VectorName:	Name of USB_DATAPUMP_APPLICATION_INIT_VECTOR structure
pSetup:	The setup function is called so that the application can do some "pre-setup", including prompting for other things to do. It is optional. The result is passed (unchanged) to all the probe and setup functions.
pFinish:	The finish function is called so that the application can do some "post-setup", including prompting for other things to do. It is optional.

3 Edge-IOCTL (Upcall) services

The following section describes the services, the Client must provide to the Protocol through the Edge- IOCTL function given when initializing the Client object using UsbPumpObject_Init() (see Section 1.2.2).

The Client shall return USBPUMP_IOCTL_RESULT_SUCCESS if it accepts the Edge-IOCTL, and USBPUMP_IOCTL_RESULT_NOT_CLAIMED if it doesn't.

3.1 Edge IOCTL Function

```
Type name :     USBPUMP_OBJECT_IOCTL_FN

Prototype :     USBPUMP_IOCTL_RESULT  OsNone_Ft900_Platform_Ioctl(

        UPLATFORM *      pPlatform,       /* Pointer to Plaform object */

        USBPUMP_IOCTL_CODE   Ioctl,       /* IOCTL code */

        CONST VOID *     pInParam,        /* Pointer to In parameter */

        VOID *           pOutParam        /* Pointer to Out parameter */

        );

Header-file : osnone_ft900_datapump.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 the 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 logical function. Note that this may be sent when there are no data-channels ready yet. This merely means that the control interface of the function has been configured and is ready to transfer data.
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 function. 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 the 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 a 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 (OS-specific driver). 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).

4 Other Considerations

[USBCDCETHER]/[USBVETHER] requires that USB Virtual Ethernet devices have unique serial (IEEE802.2 MAC address) numbers of a specific format. The USB DataPump has complete support for serial numbers, but some platform-specific code is needed to actually provide the serial number to the DataPump.

5 Contact Information

Headquarters – Singapore

Bridgetek Pte Ltd
178 Paya Lebar Road, #07-03
Singapore 409030
Tel: +65 6547 4827
Fax: +65 6841 6071

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office – Taipei, Taiwan

Bridgetek Pte Ltd, Taiwan Branch
2 Floor, No. 516, Sec. 1, Nei Hu Road, Nei Hu District
Taipei 114
Taiwan, R.O.C.
Tel: +886 (2) 8797 5691
Fax: +886 (2) 8751 9737

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Branch Office - Glasgow, United Kingdom

Bridgetek Pte. Ltd.
Unit 1, 2 Seaward Place, Centurion Business Park
Glasgow G41 1HH
United Kingdom
Tel: +44 (0) 141 429 2777
Fax: +44 (0) 141 429 2758

E-mail (Sales) sales.emea@brtichip.com
E-mail (Support) support.emea@brtchip.com

Branch Office – Vietnam

Bridgetek VietNam Company Limited
Lutaco Tower Building, 5th Floor, 173A Nguyen Van Troi,
Ward 11, Phu Nhuan District,
Ho Chi Minh City, Vietnam
Tel : 08 38453222
Fax : 08 38455222

E-mail (Sales) sales.apac@brtchip.com
E-mail (Support) support.apac@brtchip.com

Web Site

http://brtchip.com/

Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

Appendix A – References

Document References

AN_402_MCCI_USB_DataPump_UserGuide

AN_400_MCCI_USB_Resource_Compiler_UserGuide

Remote Network Driver Interface Specification (RNDIS) Protocol Specification. This specification is available at http://msdn.microsoft.com/en-us/library/ee524902(PROT.10).aspx

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

Universal Serial Bus CDC Ethernet Class Specification Overview, version 1.2. This specification is available at http://www.usb.org/developers/devclass

"Windows Hardware and Driver Central, USB Storage FAQ". This document is available at http://www.microsoft.com/whdc/connect/USB/default.mspx

Acronyms and Abbreviations

Terms	Description
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.

Appendix B – List of Tables & Figures

List of Tables

NA

List of Figures

NA

Appendix C – Revision History

Document Title:	AN_406 MCCI-USB DataPump Virtual Ethernet Protocol User Guide
Document Reference No.:	BRT_000126
Clearance No.:	BRT#096
Product Page:	http://brtchip.com/product/
Document Feedback:	Send Feedback

Revision	Changes	Date
1.0	Initial release	2017-09-13