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

MCCI USB DataPump HID Protocol User's Guide

Engineering Report 950000387
Rev. D
Date: 2011/09/30

PROPRIETARY NOTICE AND DISCLAIMER

Unless noted otherwise, this document and the information herein disclosed are proprietary to MCCI Corporation, 3520 Krums Corners Road, Ithaca, New York 14850 ("MCCI"). Any person or entity to whom this document is furnished or having possession thereof, by acceptance, assumes custody thereof and agrees that the document is given in confidence and will not be copied or reproduced in whole or in part, nor used or revealed to any person in any manner except to meet the purposes for which it was delivered. Additional rights and obligations regarding this document and its contents may be defined by a separate written agreement with MCCI, and if so, such separate written agreement shall be controlling.

The information in this document is subject to change without notice, and should not be construed as a commitment by MCCI. Although MCCI will make every effort to inform users of substantive errors, MCCI disclaims all liability for any loss or damage resulting from the use of this manual or any software described herein, including without limitation contingent, special, or incidental liability.

MCCI, TrueCard, TrueTask, MCCI Catena, and MCCI USB DataPump are registered trademarks of MCCI Corporation.

MCCI Instant RS-232, MCCI Wombat and InstallRight Pro are trademarks of MCCI Corporation.

All other trademarks and registered trademarks are owned by the respective holders of the trademarks or registered trademarks.

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	2006/08/29	Original release
Rev. B	2007/11/13	Added QueueForOutReportV2() API
Rev. C	2011/05/17	Changed document numbers to nine digit versions. DataPump 3.0 Updates
Rev. D	2011/09/30	Added source code disclaimer.

TABLE OF CONTENTS

1 Introduction 1

1.1 Related Documents 1

1.2 Glossary 1

2 Overview 2

2.1 Limitations and Cautions in Use 2

3 Implementation Details 2

3.1 Directory Structure 2

3.2 URC File Contents 3

3.3 Header files 4

3.4 Build System Considerations 4

3.5 Protocol Init Vector Contents 4

4 API Functions and Macros 5

4.1 UsbHid10_ProtocolCreate 5

4.2 UPROTO_USBHID_CONFIG_SETUP_V3() 6

5 Public Methods of UPROTO_USBHID 6

5.1 UPROTO_USBHID::QueueForOutReport() 6

5.2 UPROTO_USBHID::QueueForOutReportV2() 7

5.3 UPROTO_USBHID::QueueInReport() 9

5.4 UPROTO_USBHID::NotifyEvent() 10

6 Client Methods used by UPROTO_USBHID 10

6.1 ClientObject::Attach() 10

6.2 ClientObject::Detach() 11

6.3 ClientObject::Configure() 11

6.4 ClientObject::Unconfigure() 12

6.5 ClientObject::Suspend() 12
6.6 ClientObject::Resume() 13
6.7 ClientObject::GetReport() 13
6.8 ClientObject::SetReport() 13
6.9 ClientObject::GetIdle() 14
6.10 ClientObject::SetIdle() 14
6.11 ClientObject::SetProtocol() 15
6.12 ClientObject::GetReportDescriptor() 15
6.13 ClientObject::SetReportDescriptor() 16
6.14 ClientObject::GetPhysicalDescriptor() 16
6.15 ClientObject::SetPhysicalDescriptor() 17
6.16 ClientObject::GetMiscellaneous() 17
6.17 ClientObject::SetMiscellaneous() 18
6.18 ClientObject::EventResponse() 18
6.19 ClientObject::GetNextReport() 19
7 APIs from usbhid10.h 19
7.1 USB_HID_wValueToReportId() 19
7.2 USB_HID_wValueToReportType() 19
7.3 USB_HID_ReportTypeIdToWValue() 20
7.4 USB_HID_wValueToDuration() 20
7.5 USB_HID_DurationReportIdToWValue() 20
7.6 USB_HID_Duration_Indefinite 20
7.7 USB_HID_DurationToMillisec() 20
7.8 USB_HID_MillisecToDuration() 20
8 API Structures 21
8.1 UPROTO_USBHID 21

8.2 UPROTO_USBHID_CONFIG 21
8.2.1 UPROTO_USBHID_CONFIG_INIT_V3() 21

8.3 UPROTO_USBHID_PUBLIC_METHODS 22

8.4 UPROTO_USBHID_CLIENT_METHODS 22

9 Procedure for Implementing HID Functions 23

9.1 Use HID Descriptor Tool to create the Report Descriptor 23

9.2 Based on info from Descriptor Tool, Create HID Class Descriptor 23

9.3 Build API functions to Create Input Reports 24

9.4 Build API functions to Decode Output Reports 25

9.5 Application Integration 25

LIST OF TABLES

Table 1. HID Directory Structure 2
Table 2. Header Files 4
Table 3. Fields in `UPROTO_USBHID_CONFIG` 21

LIST OF FIGURES

Figure 1. Sample URC Code 3
Figure 2. Sample Protocol Initialization Entry 5
Figure 3. HID Descriptor Tool 23
Figure 4. Report Descriptor Example 23

1 Introduction

This document describes and specifies the HID class protocol implementation for the MCCI USB DataPump V3.0.

This document assumes familiarity with the USB-IF Human Interface Device class documents, and with the theory of HID class device design and implementation, as given for example in reference [AXEL2005].

1.1 Related Documents

[AXEL2005] *USB Complete: Everything You Need To Develop USB Peripherals* (Third Edition), Jan Axelson, 2005, Madison WI, Lakeview Research

[AXEL-USBHIDIO] *usbhidio.exe*, Visual C++ 6 HID I/O exercising program, Jan Axelson, 2005, Madison WI, Lakeview Research. Available on the web at http://www.lvr.com/hidpage.htm#MyExampleCode and http://www.lvr.com/files/usbhidio_vc6.zip

[HIDLOOP] *MCCI Hidloop User's Guide*

[USB2.0/3.0] *Universal Serial Bus Specification*, version 2.0/3.0, USB Implementers Forum, available on the web at http://www.usb.org

[USBHID1.11] *Device Class Specification for Human Interface Devices (HID),* revision 1.11, 2001, USB Implementers Forum, available on the web at http://www.usb.org/developers/hidpage

[USBHIDDT2_4] *USB HID Descriptor tool*, USB Implementers Forum, available on the web at http://www.usb.org/developers/hidpage/#Descriptor_Tool and http://www.usb.org/developers/hidpage/dt2_4.zip

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

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

1.2 Glossary

DataPump — *See* MCCI USB DataPump.

DCD — Device Controller Driver – the software component in the DataPump that manages the low-level USB device hardware. Different DCDs are used for different hardware register models.

HID — Human Interface Device

HID Protocol	The USB DataPump Module that maps an abstract HID API onto the physical transport provided by the DataPump for USB devices.
MCCI USB DataPump	MCCI's trademark for its portable USB device implementation framework.
.URC file	An input file for USBRC, specifying the desired device layout.
USB	Universal Serial Bus
USB-IF	USB Implementers 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.

2 Overview

2.1 Limitations and Cautions in Use

This implementation has one area of difficulty in its use. The programmer is responsible for making sure that the report descriptor length given in the HID descriptor in the URC file matches the actual length of the report descriptor, as implemented by the programmer-supplied code in the `GetReportDescriptor` method. No error checking is performed by the DataPump tools, build system or run time to ensure that this is so.

Although this implementation parses Get Idle and Set Idle requests, there is no built-in support for these features. Clients must capture the idle values and modulate the report submission rate according to their application requirements.

3 Implementation Details

3.1 Directory Structure

The HID protocol implementation is located in the directories shown in the following table.

Table 1. HID Directory Structure

Directory	Description
`proto/hid/i`	Header files for common use.
`proto/hid/common`	Source files implementing the protocol.

Directory	Description
app/hiddemo/hiddemo_vendor	A simple demo application for a single-function HID device that works well with the UsbHidIo program (see [AXEL-USBHIDIO]).
app/hiddemo/hiddemo_vendor2	A simple demo application similar to hiddemo_vendor. The difference is that hiddemo_vendor2 supports report ID for input/output report and support output report via interrupt pipe, while hiddemo_vendor does not. Both hiddemo_vendor and hiddemo_vendor2 demo application can be tested using the MCCI hidloop sample application (see [HIDLOOP]).
app/hiddemo/catportx	A more complex demo application that simulates a HID keyboard and HID mouse as part of a composite device.

3.2 URC File Contents

To use HID, a HID compliant interface must be added to the device's URC file, as shown below.

Figure 1. Sample URC Code

```
interface ?
          {
          class 3     %hid%
          subclass 0
          protocol 0
          # no name
          private-descriptors
              {
              raw {
                  0x21        %hid descriptor%
                  word(0x101) % hid version BCD 1.01 %
                  0x00        % not localized %
                  0x01        % only 1 HID descriptor %
                  0x22        % it's a report descriptor %
                  word(47)    % the length is 47 decimal %
                  };
              }
          endpoints
              interrupt in ? packet-size 64
                      polling-interval 10
              [interrupt out ? packet-size 64
                      polling-interval 10]
              ;
          }
```

In Figure 1, the underlined entries need to be changed to suit the customer application If the support for output report via interrupt-out endpoint is required, the second endpoint declaration (enclosed between [and]) should be used. If not, it should be removed.

3.3 Header files

Two header files are part of the external interface required to use the MCCI HID protocol implementation:

Table 2. Header Files

Name	Location	Function
`usbhid10.h`	`usbkern/i`	defines the constants from the USB HID class specification 1. 0 through 1.11.
`protohid.h`	`usbkern/proto/hid/i`	defines the externally-visible portions of the HID class implementation.

The constants defined in `usbhid10.h` are mostly self-descriptive and well documented in the header file itself. However, some useful macros and constants are listed in Section 7.

This document serves as a reference for the APIs defined by `protohid.h`.

3.4 Build System Considerations

The following entries are required in the UsbMakefile.inc that builds the application:

- `USER_CPPINCPATHS` must include the value `proto/hid/i`
- `LIBS` must include the value `protohid.$A`

These are in addition to other values required by other protocols. For example, if the application needs to use both HID and Loopback, the following entries can be added:

```
USER_CPPINCPATHS := proto/hid/i proto/loopback/i
LIBS := protohid.$A protolb.$A
```

3.5 Protocol Init Vector Contents

To incorporate support for HID into the application's runtime, the following is needed in the application's protocol init vector.

The normal usage is shown in the following figure.

Figure 2. Sample Protocol Initialization Entry

```
#include "protohid.h"
/* ... */

extern UPROTO_USBHID_CONFIG   UserSuppliedProtoConfig;

/* ... */
static
CONST USBPUMP_PROTOCOL_INIT_NODE InitNodes[] =
         {
         /* ... */
         USBPUMP_PROTOCOL_INIT_NODE_INIT_V1(
             /* dev class, subclass, proto */ -1, -1, -1,                \
             /* ifc class */ USB_bInterfaceClass_HID,                    \
             /* subclass */ 0,                                           \
             /* proto */ 0,                                              \
             /* cfg, ifc, altset */ -1, -1, -1,                          \
             /* speed */ -1,                                             \
             /* probe */ NULL,                                           \
             /* create */ UsbHid10_ProtocolCreate,                       \
             /* optional info */ (VOID *) &UserSuppliedProtoConfig       \
             ),
         /* ... */
         };
```

The above usage causes one instance of the HID protocol to be created and attached to each matching interface marked as HID class. The concrete implementation details are provided by the `USBPUMP_USBHID_CONFIG` structure `UserSuppliedProtoConfig`, which in this example is defined in another module. See `UPROTO_USBHID_CONFIG` and `UPROTO_USBHID_CONFIG_INIT_V3()` for additional information.

4 API Functions and Macros

4.1 UsbHid10_ProtocolCreate

```
USBPUMP_PROTOCOL_CREATE_FN UsbHid10_ProtocolCreate;

BOOL UsbHid10_ProtocolCreate(
         UDEVICE *pDevice,
         UINTERFACE *pInterface,
         CONST USBPUMP_PROTOCOL_INIT_NODE *pNode,
         USBPUMP_OBJECT_HEADER *pProtoInitContext
         );
```

This function is normally used in the "attach function" slot of one or more `USBPUMP_PROTOCOL_INIT_NODE` entries in the DataPump device protocol initialization

vector. `pNode->pOptionalInfo` must point to a `CONST UPROTO_USBHID_CONFIG` structure, containing information for the instance initialization. See section 3.5 for more details.

4.2 UPROTO_USBHID_CONFIG_SETUP_V3()

```
VOID UPROTO_USBHID_CONFIG_SETUP_V2(
          UPROTO_USBHID_CONFIG *pHidConfig,
          CONST TEXT *pNameOverride,
          BYTES sizeClientObject,
          CONST UPROTO_USBHID_CLIENT_METHODS *pHidClientMethods,
          USHORT sizeIntOutReportBuffer;
          USHORT sizeHostDataBuffer;
          );
```

This function-like macro is used to initialize a `UPROTO_USBHID_CONFIG` object dynamically at run time. This API may be useful in special applications that need to call `UsbHid10_ProtocolCreate()` directly, but it is not normally used. Instead, most applications will create the `UPROTO_USBHID_CONFIG` object statically at compile time using `UPROTO_USBHID_CONFIG_INIT_V3()`.

5 Public Methods of UPROTO_USBHID

The methods documented in this section are exported by the HID protocol.

5.1 UPROTO_USBHID::QueueForOutReport()

Note: This API is deprecated (supported for compatibility with old applications). Please use new API; UPROTO_USBHID::QueueForOutReportV2()

```
UPROTO_USBHID_PUBLIC_METHOD_QUEUE_FOR_OUT_REPORT_FN    QueueForOutReport;

VOID (*pHid->uhid_pPublicMethods->QueueForOutReport)(
          UPROTO_USBHID *pHid,
          UBUFQE *pQe,
          USHORT usReportTag
          );
```

NOTE: This routine must be called from DataPump context.

This routine is used asynchronously to receive outbound reports from the USB host into the buffer specified by `pQe`. According to [USBHID1.11], the host is allowed to set input reports, output reports or feature reports for a HID function via the default pipe of the device.

When the host sets a report, it is described by the wValue field of the HID Class Set Report SETUP packet. The high byte indicates the report type (input, output or feature) and the low byte indicates the report index.

To receive a report, the caller must first build a report tag that indicates the desired report, using `USB_HID_ReportTypeIdToWValue()`. The caller must then prepare a UBUFQE referencing a buffer large enough to receive the desired report, and giving a completion function that will be called when a matching report has been received.

Later, when the report is received, the UBUFQE's completion function will be called with status USTAT_OK and with uqe_ars set to the number of bytes placed into the buffer. The completion function should re-queue the UBUFQE if more reports are to be received.

Some care must be taken in constructing the report tag. If report IDs are not used, the tag should be constructed using one of the following:

```
USB_HID_ReportTypeIdToWValue(USB_HID_ReportType_Input, USB_HID_ReportID_NULL)
USB_HID_ReportTypeIdToWValue(USB_HID_ReportType_Output, USB_HID_ReportID_NULL)
USB_HID_ReportTypeIdToWValue(USB_HID_ReportType_Feature, USB_HID_ReportID_NULL)
```

Otherwise, the desired report ID must be used as the second parameter.

In all cases, the bytes written by the host are delivered to the report buffer unchanged. If report IDs are in use (as specified by the report descriptor), the client code must treat the first byte of the report buffer as a message ID; otherwise the client code must treat the first byte of the report buffer as report data.

If a report is received and no matching `UBUFQE` is found for the report, the HID protocol implementation checks to see whether a `ClientObject::SetReport()`method (section 6.8) was provided. If so, that method is invoked to process the report. Otherwise, the implementation returns an error handshake (STALL PID) to the host.

5.2 UPROTO_USBHID::QueueForOutReportV2()

```
UPROTO_USBHID_PUBLIC_METHOD_QUEUE_FOR_OUT_REPORT_V2_FN QueueForOutReportV2;

VOID (*pHid->uhid_pPublicMethods->QueueForOutReportV2)(
          UPROTO_USBHID * pHid,
          UPROTO_USBHID_OUT_REPORT_QE * pHidOutReportQe
          );
```

NOTE: This routine must be called from DataPump context.

This routine is used asynchronously to receive outbound reports, both via default pipe and optional interrupt pipe, from the USB host into the buffer specified by `pHidOutReportQe->pBuffer`. According to [USBHID1.11], the host is allowed to set input reports, output reports or feature reports for a HID function via the default pipe of the device and optionally send output

report to a HID function via the interrupt pipe if interrupt out endpoint is declared in HID interface descriptor.

When the host sets a report via default pipe of the device, it is described in the wValue field of the HID Class Set Report SETUP packet. The high byte indicates the report type (input, output or feature) and the low byte indicates the report ID if report ID value is declared in the corresponding HID report descriptor item, otherwise low byte is set to zero. The first byte of report data being transferred in data stage of control transfer is report ID field if report ID is used. If not, report data start with real report data contents without report ID field.

When the host sends output report via interrupt pipe of the HID interface (input and feature report is not supported via interrupt pipe according to [USBHID1.11]), report ID occupies the first byte of the output report if the device use report ID as the case of output report via default pipe.

To receive a report, the caller must first build a UPROTO_USBHID_OUT_REPORT_QE with each fields initialized with proper values using `UPROTO_USBHID_OUT_REPORT_QE_SETUP_V1()` macro.

```
UPROTO_USBHID_OUT_REPORT_QE_SETUP_V1 (
          pHidOutReportQe, /* UPROTO_USBHID_OUT_REPORT_QE type queue element */
          pBuffer,         /* IN: buffer pointer */
          nBuffer,         /* IN: buffer size */
          wReportTag,      /* IN: report tag */
          pDoneFn,         /* IN: done function */
          pDoneInfo        /* IN: done function context */
          };
```

`pBuffer` should reference application buffer large enough to receive the desired report.

`wReportTag` indicates the report that application requests to receive, can be set by `USB_HID_ReportTypeIdToWValue()`. Some care must be taken in constructing `wReportTag`. If the report ID is not used, the tag should be constructed like below:

```
wReportTag = USB_HID_ReportTypeIdToWValue(USB_HID_ReportType_Output, 0);
```

Otherwise, the desired report ID must be used as the second parameter:

```
wReportTag = USB_HID_ReportTypeIdToWValue(USB_HID_ReportType_Input, 1);
```

`pDoneFn` is a function type and would get called when report is received; with `pHidOutReportQe->nReceived` set to the received report data size and with `pHidOutReportQe->Status` set to the `Status` parameter value (`USTAT_OK` for normal success, others for failure). Client application should check if the `Status` value is acceptable for its own purpose. If so, should return TRUE, otherwise FALSE which would make the device return STALL_PID to the host.

```
BOOL hidvendor_OutReportDone(
          UPROTO_USBHID_OUT_REPORT_QE *    pHidOutReportQe,
          VOID *                           pDoneInfo,
          USTAT                            Status
          );
```

In all cases, the bytes transferred from the host are delivered to the report buffer unchanged. If report ID is in use (as specified by the report descriptor), the client code must treat the first byte of the report buffer as a report id; otherwise the client code must treat the first byte of the report buffer as report data content.

If a report is received but no matching UPROTO_USBHID_OUT_REPORT_QE is found for the report, the HID protocol implementation checks to see whether a `ClientObject::SetReport()` method (section 6.8) was provided. If so, that method is invoked to process the report. Otherwise, the implementation returns an error handshake (STALL PID) to the host.

5.3 UPROTO_USBHID::QueueInReport()

```
UPROTO_USBHID_PUBLIC_METHOD_QUEUE_IN_REPORT_FN     QueueInReport;

VOID (*pHid->uhid_pPublicMethods->QueueInReport)(
          UPROTO_USBHID *pHid,
          UBUFQE *pQe
          );
```

NOTE: This routine must be called from DataPump context.

This routine is used asynchronously, to submit a report to be transmitted to the host over the HID function's Interrupt IN pipe.

The client should specify the buffer size and length in the UBUFQE, and should specify a completion function to be called when the data has been transferred.

On completion, the client must check the completion status – if it's not USTAT_OK, then it is likely that an unplug or configuration change event has occurred, and resubmitting the I/O is likely to fail until the host re-enables the device.

Note that `UPROTO_USBHID::GetNextReport()` will also be called whenever the input queue for the Interrupt IN pipe is empty.

Clients must be careful not to access `pQe` or resubmit it while it is still in use. The synchronization model of the DataPump allows client code that is running in DataPump context to check the value of `pQe->uqe_status` for this purpose. If the value is `USTAT_BUSY`, then the `UBUFQE` is still in use by the HID protocol or lower layers. Otherwise, the `UBUFQE` has been completed and may be reused.

5.4 UPROTO_USBHID::NotifyEvent()

```
UPROTO_USBHID_PUBLIC_METHOD_NOTIFY_EVENT_FN        NotifyEvent;

VOID (*pHid->uhid_pPublicMethods->NotifyEvent)(
          UPROTO_USBHID *pHid
          );
```

NOTE: This routine may be called in arbitrary context; it synchronizes to the DataPump

---

This routine is called by clients who are not running in DataPump context, and who need to be called back in DataPump context.

An event is queued to the DataPump. Later, when the event is processed, the following events take place in DataPump context:

1. The HID protocol calls the `ClientObject::EventResponse()` method, if one was specified.

2. The HID protocol checks the status of the input pipe queue. If the queue is empty and another report may be queued by the client, then the HID protocol calls the `ClientObject::GetNextReport()`method (if one was specified).

3. In all cases, the HID protocol attempts to submit the next I/O for the Interrupt IN pipe.

6 Client Methods used by UPROTO_USBHID

In the following section, we use a pseudo-C++ syntax to describe the object methods that the user must write and supply for the use of the HID protocol code. These pointers are provided by filling in a `UPROTO_USBHID_CLIENT_METHODS` table. This is normally done at compile time using a `UPROTO_USBHID_CLIENT_METHODS_INIT_V1()` macro.

All of the methods listed in this section will be called in DataPump context.

6.1 ClientObject::Attach()

```
UPROTO_USBHID_CLIENT_METHOD_ATTACH_FN      pHid->pClientMethods->Attach;

BOOL ClientObject::Attach(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called when the HID protocol implementation has determined a client object and method table to be used for this instance. Normally this happens while `UsbHid10_ProtocolCreate()` is running, but in a future upgrade we may choose to allow this to be deferred until a client opens the HID instance.

If this function returns `FALSE`, then the attach operation is cancelled (and ClientObject::Detach() will be called).

6.2 ClientObject::Detach()

```
PUPROTO_USBHID_CLIENT_METHOD_DETACH_FN     pHid->pClientMethods->Detach;

VOID ClientObject::Detach(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called when the HID protocol implementation is tearing down the protocol connection. In the current implementation, this can only happen when some kind of error occurs during `UsbHid10_ProtocolCreate()` processing, after ClientObject::Attach() has been called.

6.3 ClientObject::Configure()

```
PUPROTO_USBHID_CLIENT_METHOD_CONFIGURE_FN      pHid->pClientMethods->Configure;

VOID ClientObject::Configure(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENT Why
          );
```

This method is called to notify the client that the underlying USB transport has just been activated, typically by a SetConfiguration command (hence the name). The parameter `Why` indicates the exact reason. The values for `UEVENT` are documented in `usbkern/i/ueventnode.h`. Normally, `Why` will be set to `UEVENT_CONFIG_SET`.

The possible values of `Why` are:

UEVENT_CONFIG_SET	0
UEVENT_CONFIG_UNSET	1
UEVENT_IFC_SET	2
UEVENT_IFC_UNSET	3
UEVENT_FEATURE	4
UEVENT_CONTROL	5
UEVENT_SUSPEND	6
UEVENT_RESUME	7

UEVENT_RESET	8
UEVENT_SETADDR	9
UEVENT_CONTROL_PRE	10
UEVENT_INTLOAD	11
UEVENT_GETDEVSTATUS	12
UEVENT_GETIFCSTATUS	13
UEVENT_GETEPSTATUS	14
UEVENT_SETADDR_EXEC	15
UEVENT_DATAPLANE	16
UEVENT_ATTACH	17
UEVENT_DETACH	18
UEVENT_PLATFORM_EXTENSION	19
UEVENT_L1_SLEEP	20
UEVENT_CABLE	21
UEVENT_NOCABLE	22

Please note that some Windows components will configure HID class interfaces, and start sending IN tokens, long before they are actually ready to receive reports on the interrupt pipe.

6.4 ClientObject::Unconfigure()

```
PUPROTO_USBHID_CLIENT_METHOD_UNCONFIGURE_FN    pHid->pClientMethods->Unconfigure;

VOID ClientObject::Unconfigure(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENT Why
          );
```

This method is called to notify the client that the underlying USB transport has just been deactivated. This may happen for any number of reasons. The parameter `Why` indicates the exact reason. The values for `UEVENT` are documented in `usbkern/i/ueventnode.h`.

Please note that on some platforms it is very hard to distinguish between a simple USB suspend and a cable unplug. The DataPump cannot send this message in response to cable unplug unless it gets an unambiguous indication from the device controller driver (DCD).

6.5 ClientObject::Suspend()

```
PUPROTO_USBHID_CLIENT_METHOD_SUSPEND_FN        pHid->pClientMethods->Suspend;

VOID ClientObject::Suspend(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called when a USB Suspend is detected for the device containing this HID function.

6.6 ClientObject::Resume()

```
PUPROTO_USBHID_CLIENT_METHOD_RESUME_FN     pHid->pClientMethods->Resume;

VOID ClientObject::Resume(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called when a USB Resume is detected for the device containing this HID function.

6.7 ClientObject::GetReport()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN      pHid->pClientMethods->GetReport;

BOOL ClientObject::GetReport(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Get Report request is received over the default pipe. (It is not called for reading reports over the Interrupt IN pipe.)

The client must decode the report tag given in `pSetup->uec_setup.ucp_wValue`, and must return the proper report using UsbDeviceReply(). See the function `hidvendor_GetReport()` in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c` for an example.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

6.8 ClientObject::SetReport()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN      pHid->pClientMethods->SetReport;

BOOL ClientObject::SetReport(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Set Report request is received over the default pipe. (It is not called for reading reports over the Interrupt IN pipe.)

The client must decode the report tag given in `pSetup->uec_setup.ucp_wValue`, and must return the proper report using `UsbDeviceReply()`. For example, please refer to the function `hidvendor_GetReport()` in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c`

This routine is only called if there was no matching `UBUFQE` submitted via the `UPROTO_USBHID::QueueForOutReport()` method.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.9 ClientObject::GetIdle()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->GetIdle;

BOOL ClientObject::GetIdle(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Get Idle request is received over the default pipe.

The client must decode the report tag given in `pSetup->uec_setup.ucp_wValue`, and must return the proper idle value to the host using `UsbDeviceReply()`.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.10 ClientObject::SetIdle()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->SetIdle;

BOOL ClientObject::SetIdle(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Set Idle request is received over the default pipe.

The client must decode the report tag given in `pSetup->uec_setup.ucp_wValue`, and must store the time and change the idle behavior of the device.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.11 ClientObject::SetProtocol()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->SetProtocol;

BOOL ClientObject::SetProtocol(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Set Protocol request is received over the default pipe.

The client must decode the protocol given in `pSetup->uec_setup.ucp_wValue`, and must change the behavior of the device as appropriate.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. It will never be called unless the bit `UHIDFLAG_SUPPORTBOOT` is set in the flags word of the UPROTO_USBHID. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.12 ClientObject::GetReportDescriptor()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->GetReportDescriptor;

BOOL ClientObject::GetReportDescriptor(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Get Report Descriptor request is received over the default pipe.

The client must decode the report selector given in `pSetup->uec_setup.ucp_wValue`, and must return the proper report descriptor to the host using `UsbDeviceReply()`. For an example, please refer to the function `hidvendor_GetReport()` in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c`

For proper operation, the client must implement this operation for every descriptor that is mentioned in the HID class descriptor in the USBRC input file for this function.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.13 ClientObject::SetReportDescriptor()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->SetReportDescriptor;

BOOL ClientObject::SetReportDescriptor(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Set Report Descriptor request is received over the default pipe.

Set Report Descriptor is normally not implemented, and so this method function is normally omitted. However, if the client wishes to support Set Report Descriptor, the client must decode the report selector given in `pSetup->uec_setup.ucp_wValue`, and must submit a UBUFQE to collect the data from the host. This is similar to the implementation of other host-to-device SETUP commands with data.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.14 ClientObject::GetPhysicalDescriptor()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->GetPhysicalDescriptor;

BOOL ClientObject::GetPhysicalDescriptor(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Get Physical Descriptor request is received over the default pipe.

Physical descriptors are not commonly used. However if the client wishes to implement them, appropriate information must be added to the URC file. Then code must be added to the client to decode the descriptor selector given in `pSetup->uec_setup.ucp_wValue`, and to return the proper physical descriptor to the host using `UsbDeviceReply()`.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.15 ClientObject::SetPhysicalDescriptor()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->SetPhysicalDescriptor;

BOOL ClientObject::SetPhysicalDescriptor(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when a HID class Set Physical Descriptor request is received over the default pipe.

Physical descriptors are rarely used, and Set Physical Descriptor is even more uncommonly used; so this method function is normally omitted. However, if the client wishes to support Set Physical Descriptor, the client must decode the report selector given in `pSetup->uec_setup.ucp_wValue`, and must submit a `UBUFQE` to collect the data from the host. This is similar to the implementation of other host-to-device SETUP commands with data.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.16 ClientObject::GetMiscellaneous()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->GetMiscellaneous;

BOOL ClientObject::GetMiscellaneous(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when an unrecognized HID class Get request is received over the default pipe.

The client must decode the setup packet given in `pSetup->uec_setup`, and must return any results to the host using `UsbDeviceReply()`.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.17 ClientObject::SetMiscellaneous()

```
PUPROTO_USBHID_CLIENT_METHOD_SETUP_FN     pHid->pClientMethods->SetMiscellaneous;

BOOL ClientObject::GetMiscellaneous(
          VOID *pClientObject,
          UPROTO_USBHID *pHid,
          UEVENTSETUP *pSetup
          );
```

This function is called when an unrecognized HID class Set request is received over the default pipe.

The client must decode the setup packet given in `pSetup->uec_setup`, and must (if necessary) submit a `UBUFQE` to the default-out pipe to collect the data from the host. This is similar to the implementation of other host-to-device SETUP commands with data.

This function should return `FALSE` if an Error Handshake (STALL PID) is to be sent to the host. Otherwise it should return `TRUE`.

This function is optional. If not provided, `UPROTO_USBHID` will substitute a function that always returns `FALSE`, which will cause an error handshake to be returned to the host.

6.18 ClientObject::EventResponse()

```
PUPROTO_USBHID_CLIENT_METHOD_EVENT_RESPONSE_FN pHid->pClientMethods->EventResponse;

VOID ClientObject::EventResponse(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called in DataPump context, in response to a previous `UPROTO_USBHID::NotifyEvent()` call from the client.

The client may take any actions desired. This function is optional. See Section 5.4 for more information.

6.19 ClientObject::GetNextReport()

```
PUPROTO_USBHID_CLIENT_METHOD_GET_NEXT_REPORT_FN pHid->pClientMethods->GetNextReport;

VOID ClientObject::GetNextReport(
          VOID *pClientObject,
          UPROTO_USBHID *pHid
          );
```

This function is called in DataPump context whenever the UPROTO_USBHID implementation determines that it's appropriate for the client to send more reports to the host over the Interrupt IN pipe using the `UPROTO_USBHID::QueueInReport()` mechanism.

This function is optional. If not provided, the implementation behaves as if an empty function had been provided.

The implementation calls this function (if appropriate) after delivering a `ClientObject:Configured()` message, after successfully sending a previous report to the host, and while doing deferred processing for `UPROTO_USBHID::NotifyEvent()`.

7 APIs from usbhid10.h

7.1 USB_HID_wValueToReportId()

GET/SET reports use wValue high/low to encode report id.

```
UCHAR USB_HID_wValueToReportId(
          ARG_USHORT wValue
          );
```

7.2 USB_HID_wValueToReportType()

GET/SET reports use wValue high/low to encode report type.

```
UCHAR USB_HID_wValueToReportType(
          ARG_USHORT wValue
          );
```

7.3 USB_HID_ReportTypeIdToWValue()

GET/SET reports use report type/id to encode wValue.

```
USHORT USB_HID_ReportTypeIdToWValue(
          ARG_UCHAR Type,
          ARG_UCHAR Id
          );
```

7.4 USB_HID_wValueToDuration()

```
UCHAR USB_HID_wValueToDuration(
          ARG_USHORT wValue
          );
```

7.5 USB_HID_DurationReportIdToWValue()

```
USHORT USB_HID_DurationReportIdToWValue(
          ARG_UCHAR Duration,
          ARG_UCHAR Id
          );
```

7.6 USB_HID_Duration_Indefinite

```
USB_HID_Duration_Indefinite
```

7.7 USB_HID_DurationToMillisec()

```
INT USB_HID_DurationToMillisec(
          INT Duration
          );
```

7.8 USB_HID_MillisecToDuration()

```
INT USB_HID_MillisecToDuration(
          INT Millisec
          );
```

8 API Structures

8.1 UPROTO_USBHID

Although this object is publicly defined in `protohid.h`, only a few fields are intended to be used by the client.

8.2 UPROTO_USBHID_CONFIG

This structure provides configuration information to `UsbPumpProtoHid_ProtocolCreate()`. It has the following entries.

Table 3. Fields in `UPROTO_USBHID_CONFIG`

Field	Description
`CONST TEXT *pNameOverride;`	If given, provides the name for this `UPROTO_USBHID` instance. Normally the name is generated using `UPROTO_USBHID_DERIVED_NAME("…")` to ensure that the name is formatted with a consistent suffix. If `NULL`, the name `UPROTO_USBHID_NAME` is used by default.
`BYTES sizeClientObject;`	The desired size of the client object. If zero, no client object is created during initialization.
`CONST UPROTO_USBHID_CLIENT_METHODS *pClientObjectMethodTable;`	Pointer to the table of method functions associated with the client object. By storing these externally, the layout of the client object is made completely opaque to the `UPROTO_USBHID` implementation.
`USHORT sizeIntOutReportBuffer`	Buffer size to be allocated at protocol init time and used to receive out-report via interrupt-out pipe. But buffer is allocated only if HID interface has interrupt-out endpoint declared.
`USHORT sizeHostDataBuffer`	Host data buffer size

8.2.1 UPROTO_USBHID_CONFIG_INIT_V3()

This macro is used to generate compile-time initialization for a UPROTO_USBHID_CONFIG object in a forward-compatible way. It's normally used as follows:

```
CONST UPROTO_USBHID_CONFIG MyProtoConfig =
         UPROTO_USBHID_CONFIG_INIT_V3(
             /* name */ UPROTO_USBHID_DERIVED_NAME("my.hid"),
             /* sizeClientObject */ sizeof(MY_CLIENT_OBJECT),
             /* methods */ &MyClientMethodTable,
             /* sizeIntOutReportBuffer */ 128,
```

```
            /* sizeHostDataBuffer*/ 64
            );
```

If the configuration structure layout changes in the future, MCCI will create a _V4() macro that initializes the new format of the structure, and will revise the _V3() macro to call the _V4() macro with suitable default values for any new parameters.

8.3 UPROTO_USBHID_PUBLIC_METHODS

This structure is supplied by the implementation of UPROTO_USBHID, and provides method functions used by the clients to effect operations on the UPROTO_USBHID object.

8.4 UPROTO_USBHID_CLIENT_METHODS

This structure is supplied by the client of UPROTO_USBHID, and provides method functions used by the UPROTO_USBHID object to send notifications to the client instance object.

Normally, the functions are all declared (even the ones that are not in use) using:

UPROTO_USBHID_CLIENT_METHODS_DECLARE_FNS(MyPrefix);

This will generate prototypes and names for all the possible method functions, for example MyPrefix_Attach(), MyPrefix_Detach(), and so forth.

Then the method table is initialized as shown in the following example:

```
CONST UPROTO_USBHID_CLIENT_METHODS MyPrefix_switch =
         UPROTO_USBHID_CLIENT_METHODS_INIT_V1(
             MyPrefix_Attach,
             MyPrefix_Detach,
             MyPrefix_Configure,
             MyPrefix_Unconfigure,
             /* MyPrefix_Suspend */ NULL,
             /* MyPrefix_Resume */ NULL,
             MyPrefix_GetNextReport,
             MyPrefix_GetReport,
             /* MyPrefix_SetReport */ NULL,
             /* MyPrefix_GetIdle */ NULL,
             /* MyPrefix_SetIdle */ NULL,
             /* MyPrefix_SetProtocol */ NULL,
             MyPrefix_GetReportDescriptor,
             /* MyPrefix_SetReportDescriptor */ NULL,
             /* MyPrefix_GetPhysicalDescriptor */ NULL,
             /* MyPrefix_SetPhysicalDescriptor */ NULL,
             /* MyPrefix_GetMiscellaneous */ NULL,
             /* MyPrefix_SetMiscellaneous */ NULL,
             /* MyPrefix_EventResponse -- NULL means use GetNextReport */ NULL
             );
```

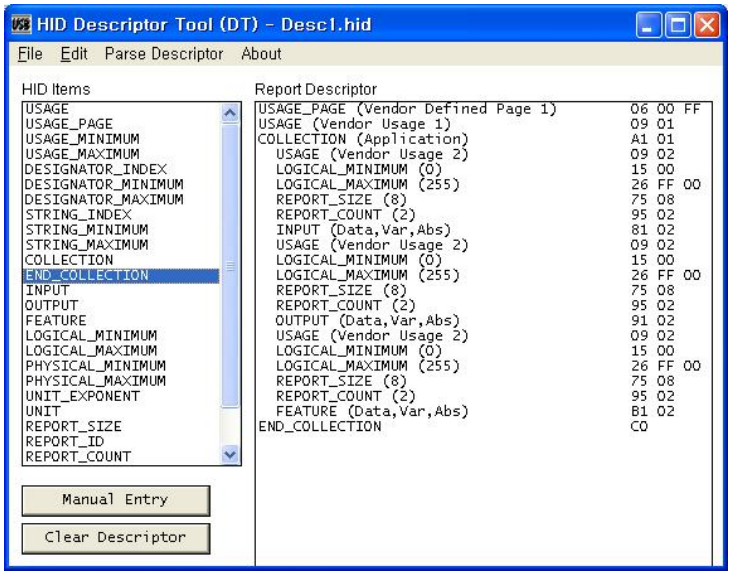
9 Procedure for Implementing HID Functions

This section outlines a step by step procedure for creating a new HID function from scratch

9.1 Use HID Descriptor Tool to create the Report Descriptor

Please refer to [USBHIDDT2_4] and download HID Descriptor Tool, this helps you to create report descriptor symbolically, letting you ignore bit-values that you have to be careful about in creating report descriptor, and then you can check if there's logical error in report descriptor you created with this tool.

Figure 3. HID Descriptor Tool



9.2 Based on info from Descriptor Tool, Create HID Class Descriptor

Below is the report descriptor derived from the previous step.

This is defined in hiddemo_vendor sample application which receive 2 bytes output report, and then send it back to the host via input report (in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_reports.c`).

Figure 4. Report Descriptor Example

```
ROM UCHAR VendorReportDescriptor[] =
          {
          0x06, 0xA0, 0xFF,    /* Usage page (vendor defined) */
          0x09, 0x01,      /* Usage ID (vendor defined) */
          0xA1, 0x01,      /* Collection (application) */

          /* The Input report */
          0x09, 0x03,      /* Usage ID - vendor defined */
```

```
        0x15, 0x00,      /* Logical Minimum (0) */
        0x26, 0xFF, 0x00,    /* Logical Maximum (255) */
        0x75, 0x08,      /* Report Size (8 bits) */
        0x95, 0x02,      /* Report Count (2 fields) */
        0x81, 0x02,      /* Input (Data, Variable, Absolute) */

        /* The Output report */
        0x09, 0x04,      /* Usage ID - vendor defined */
        0x15, 0x00,      /* Logical Minimum (0) */
        0x26, 0xFF, 0x00,    /* Logical Maximum (255) */
        0x75, 0x08,      /* Report Size (8 bits) */
        0x95, 0x02,      /* Report Count (2 fields) */
        0x91, 0x02,          /* Output (Data, Variable, Absolute) */

        /* The Feature report */
        0x09, 0x05,      /* Usage ID - vendor defined */
        0x15, 0x00,      /* Logical Minimum (0) */
        0x26, 0xFF, 0x00,    /* Logical Maximum (255) */
        0x75, 0x08,      /* Report Size (8 bits) */
        0x95, 0x02,      /* Report Count (2 fields) */
        0xB1, 0x02,      /* Feature (Data, Variable, Absolute) */

        0xC0         /* end collection */
        };
```

Client application should register ClientObject::GetReportDescriptor() method in its client method table and implement the logic which transfers this report descriptor to the host.

Please refer to hidvendor_GetReportDescriptor() in file
`usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c.`

9.3 Build API functions to Create Input Reports

The USB Host has two ways to retrieve the input report from the device. The first one is via default pipe and the second one is via interrupt IN pipe.

If the device supports input report transfer via default pipe, the client application should register `ClientObject::GetReport()` method in its client method table and implement the logic which transfers input reports to the host by calling `UsbDeviceReply()` DataPump API.

Please refer to `hidvendor_GetReport()` in the file
`usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c.`

If the device supports input report transfer via interrupt IN pipe, the client application should register `ClientObject::GetNextReport()` method in its client method table and implement the logic which transfers input report to the host by calling `UPROTO_USBHID::QueueInReport()` HID API.

Please refer to `hidvendor_GetNextReport()` in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c`.

9.4 Build API functions to Decode Output Reports

As described in the overview, output reports can be transferred to the device via default pipe of the device or interrupt out pipe declared in HID interface descriptor. To receive output reports, client application should queue `UPROTO_USBHID_OUT_REPORT_QE` by calling `UPROTO_USBHID::QueueForOutReportV2()` HID API.

The client should keep `UPROTO_USBHID_OUT_REPORT_QE` queued on HID protocol module in order not to miss output reports coming from the host. This logic can usually be implemented by queuing `UPROTO_USBHID_OUT_REPORT_QE` when HID interface gets configured (when `ClientObject::Configure()` client method is called by DataPump) and then queuing `UPROTO_USBHID_OUT_REPORT_QE` again in the `UPROTO_USBHID_OUT_REPORT_QE_DONE_FN`. Please refer to `hidvendor_Configure()` and `hidvendor_OutReportDone()` in file `usbkern/app/hiddemo/hiddemo_vendor/hiddemo_vendor_implementation.c`.

9.5 Application Integration

To make the methods above available to the DataPump, the client should put the client method table reference into `UPROTO_USBHID_CONFIG` object and pass it to protocol init vector as described in section 3.5 Protocol Init Vector Contents.