USB Power Delivery Analyzer

The USB Power Delivery Analyzer allows developers to interface a host PC to a downstream embedded system environment and non-intrusively monitors power delivery data in real time as it appears on the bus.

USB Power Delivery Analyzer Features

- Power Delivery Interface
  - Non-intrusive power delivery monitoring
  - Transparent interposing on a USB Type-C connection
  - Monitoring Control Channel lines CC1 and CC2
  - Monitoring VBUS and VCONN voltages and currents
- Software
  - Windows, Linux, and Mac OS X compatible
  - Data Center

Supported products:

USB Power Delivery Analyzer
User Manual v1.20.002
December 14, 2018

1 General Overview

1.1 Revision History

1.1.1 Changes in version 1.20

Initial API release.

1.1.2 Changes in version 1.00

Initial revision.

1.2 General Description

The USB Power Delivery Analyzer (PD analyzer) non-intrusively monitors power delivery data on Control Channel lines CC1 and CC2 through USB Type C connection. The PD analyzer acts as USB Pass Through for Super-Speed 5/10 Gbps (USB 3.1 Gen 1/2), Hi-Speed 480 Mbps, Full-Speed 12 Mbps, and Low-Speed 1.5 Mbps. Total Phase worked closely with Google to bring this Chromium project design (code name "Twinkie") to market. The Total Phase USB PD analyzer goes beyond the original Twinkie design, offering improved firmware and the ability to use it with Total Phase Data Center Software. The PD analyzer captures and displays data using Data Center Software. The PD analyzer is manufactured in Total Phase US-based, ISO9001 certified facility. For more detail, please visit http://www.totalphase.com/products/data-center/.

2 Hardware Specification

2.1 Connector Description

PD analyzer connectors are described in figure 1. PD analyzer analysis USB micro B receptacle (left connector in Figure 1) should be connected to analysis computer. PD analyzer analysis USB micro B receptacle can be connected for example to analysis computer USB type A receptacle through micro B male to type A male cable. PD analyzer target USB type C plug (top connector in Figure 1) should be connected to target USB type C receptacle. PD analyzer target USB type C receptacle (bottom connector in Figure 1) should be connected to target USB type C plug. The included cable is 6" USB micro B to USB type A.

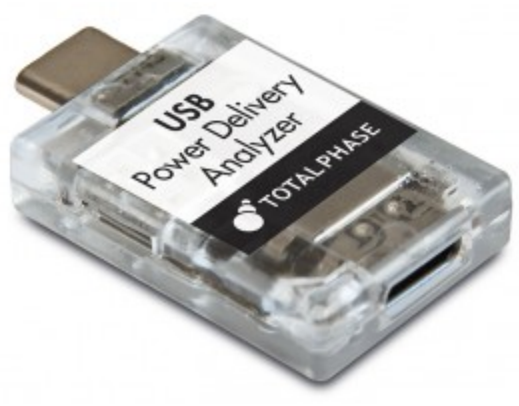

Figure 1 *: PD Analyzer Connectors*

VBus supports maximum is 20 V and 5 A.

PD analyzer block diagram is described in Figure 2.

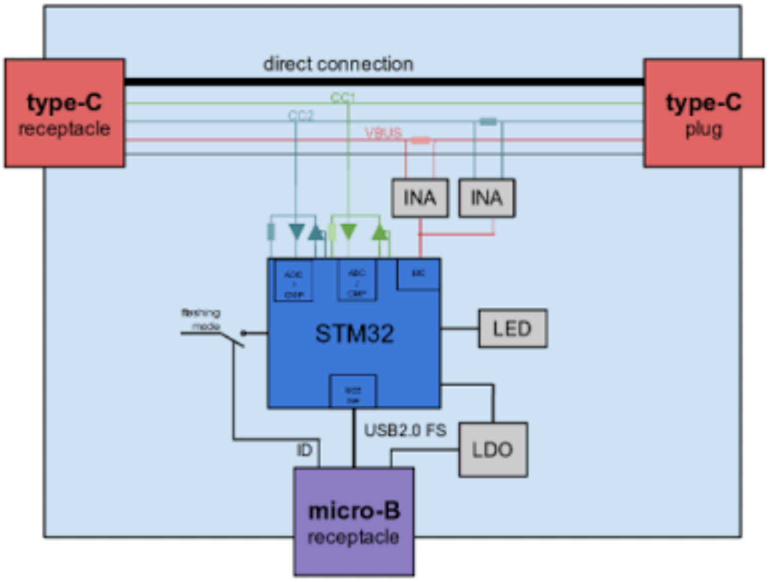

Figure 2 *: PD Analyzer Block Diagram*

2.2 Known USB Power Delivery Limitation

On computers with a highly loaded USB sub system, performance may be degraded and data may be lost between the Power Delivery Analyzer and the analysis computer.

2.3 Physical Specifications

Dimension: W x D x L: 25.4 mm x 38.1 mm x 6.4 mm (1.00 in x 1.5 in, 0.25 in)

Weight: 42 g (1.5 oz)

3 Software

3.1 Compatibility

3.1.1 Overview

The PD analyzer GUI, Data Center, is offered as a 32-bit or 64-bit application. The specific compatibility for each operating system is discussed below.

3.1.2 Windows Compatibility

The PD analyzer GUI, Data Center, is compatible with 32-bit and 64-bit versions of Windows 7/8/8.1/10. The software is provided as a 32-bit or 64-bit application.

3.1.3 Linux Compatibility

The PD analyzer GUI, Data Center, is compatible with 32-bit and 64-bit standard distributions of Linux with kernel 2.6 and integrated USB support including: Red Hat, Ubuntu, Fedora, and SuSE. The software is provided as a 32-bit or 64-bit application. When using the 32-bit library on a 64-bit distribution, the appropriate 32-bit system libraries are also required. When using either the 64-bit library or 32 bit library, the appropriate system libraries are also required.

3.1.4 Mac OS X Compatibility

The PD analyzer GUI, Data Center, is compatible with 32-bit and 64-bit Intel versions of Mac OS X 10.7 Lion, 10.8 Mountain Lion, 10.9 Mavericks, 10.10 Yosemite, and 10.11 El Capitan. The software is provided as a 32-bit or 64-bit application.

3.2 USB Driver

3.2.1 Windows USB Driver

Windows 8/8.1/10

Plug PD analyzer to analysis computer, wait till installation is done, and use Data Center. PD analyzer should show up as Total Phase PD analyzer in Device Manager under Universal Serial Bus devices section.

Windows 7

Plug PD analyzer to analysis computer, wait till installation is done, and verify that PD analyzer shows up as Total Phase PD analyzer or WinUSB Device in Device Manager under Universal Serial Bus devices section. If PD analyzer shows up as Total Phase PD analyzer or WinUSB Device in Device Manager under Universal Serial Bus devices section, then use Data Center. If PD analyzer shows up as Total Phase PD Analyzer under Other devices section, then follow the steps below

1. Go to Control Panel window -> Click System tab -> Click Windows Update icon -> Click Install updates icon -> Mark 'I accept the license terms' -> Click Finish icon -> Wait till Windows Update done.
2. Go to Control Panel window -> Click System tab -> Click Advanced system settings icon -> Click Hardware icon -> Click Device Installation Setting icon -> Mark 'No, Let me choose what I do' and 'Install driver software from Windows Update if it is not found on my computer' -> Click Save Changes icon -> Click OK
3. Unplug and plug PD analyzer from analysis computer, and wait till Driver software is installed
4. If PD analyzer shows up in Device Manager as WinUsb Device under Universal Serial Bus devices section, then your PD analyzer is installed successfully on the analysis computer, and you can use Data Center.
5. If PD analyzer shows up in Device Manager under Other devices section, then download and install USB driver for Windows v2.13 unsigned beta from wwww.totalphase.com on your analysis computer as described in section 'Windows USB Driver v2.13 Unsigned Beta' below.
6. Verify that PD analyzer shows up in Device Manager as WinUSB Device under Universal Serial Bus devices section, and use Data Center.

Windows USB Driver v2.13 Unsigned Beta

To install the appropriate USB communication driver under Windows, use Total Phase USB Driver v2.13 unsigned beta before plugging in any device. The driver can be found in the Total Phase website www.totalphase.com. After the driver has been installed, plugging in a PD analyzer for the first time will cause the PD analyzer to be installed and associated with the correct driver. The following steps describe the feedback the user should receive from Windows after a PD analyzer is plugged into a system for the first time:

1. A notification bubble will pop up from the system tray and state that Windows is "installing device driver software."
2. When the installation is complete, the notification bubble will again pop up and state that the "device driver software installed successfully."

To confirm that the device was correctly installed, check that the device appears in the "Device Manager." For Windows 7/8/8.1/10, to navigate to the "Device Manager"screen, select "Control Panel | Hardware and Sound | Device Manager". The PD analyzer should appear under the Universal Serial Bus Controllers section.

Windows USB Driver v2.13 Unsigned Beta Removal

The USB communication driver can be removed from the operating system by using the Windows program removal utility. Instructions for using this utility can be found below. Alternatively, the Uninstall option found in the driver installer can also be used to remove the driver from the system.

NOTE: It is critical to remove all Total Phase devices from your system before you remove the USB drivers.

1. Select "Control Panel | Uninstall a program"
2. Right-click on "Total Phase USB Driver" and select "Uninstall/Change"
3. Follow the instructions in the uninstaller

3.2.2 Linux USB Driver

The PD analyzer communications layer under Linux does not require a specific kernel driver to operate. However, the user must ensure independently that the libusb library is installed on the system since the PD analyzer library is dynamically linked to libusb.

Most modern Linux distributions use the udev subsystem to help manipulate the permissions of various system devices. This is the preferred way to support access to the PD analyzer such that the device is accessible by all of the users on the system upon device plug-in.

For legacy systems, there are two different ways to access the PD analyzer, through USB hotplug or by mounting the entire USB filesystem as world writable. Both require that /proc/bus/usb is mounted on the system which is the case on most standard distributions.

UDEV

Support for udev requires a single configuration file that is available on the Total Phase website www.totalphase.com for download. This file is 99-totalphase.rules. Please follow the following steps to enable the appropriate permissions for the PD analyzer.

1. As superuser, unpack 99-totalphase.rules to /etc/udev/rules.d
2. `chmod 644 /etc/udev/rules.d/99-totalphase.rules`
3. Unplug and replug your PD analyzers

USB Hotplug

USB hotplug requires two configuration files which are available on the Total Phase website www.totalphase.com for download. These files are: pd and pd.usermap. Please follow the following steps to enable hotplugging.

1. As superuser, unpack pd and pd.usermap to /etc/hotplug/usb
2. `chmod 755 /etc/hotplug/usb/pd`
3. `chmod 644 /etc/hotplug/usb/pd.usermap`
4. Unplug and replug your PD analyzers
5. Set the environment variable USB_DEVFS_PATH to /proc/bus/usb

World-Writable USB Filesystem

Finally, here is a last-ditch method for configuring your Linux system in the event that your distribution does not have udev or hotplug capabilities. The following procedure is not necessary if you were able to exercise the steps in the previous subsections.

Often, the /proc/bus/usb directory is mounted with read-write permissions for root and read-only permissions for all other users. If an non-privileged user wishes to use the PD analyzer and software, one must ensure that /proc/bus/usb is mounted with read-write permissions for all users. The following steps can help setup the correct permissions. Please note that these steps will make the entire USB filesystem world writable.

1. Check the current permissions by executing the following command:
   `ls -al /proc/bus/usb/001`
2. If the contents of that directory are only writable by root, proceed with the remaining steps outlined below.
3. Add the following line to the `/etc/fstab` file:

   ```
   none /proc/bus/usb usbfs defaults,devmode=0666 0 0
   ```

4. Unmount the `/proc/bus/usb` directory using `umount`
5. Remount the `/proc/bus/usb` directory using `mount`
6. Repeat step 1. Now the contents of that directory should be writable by all users.
7. Set the environment variable `USB_DEVFS_PATH` to `/proc/bus/usb`

3.2.3 Mac OS X USB Driver

The PD analyzer communications layer under Mac OS X does not require a specific kernel driver to operate. Mac OS X versions 10.7 Lion, 10.8 Mountain Lion, 10.9 Mavericks, 10.10 Yosemite, and 10.11 El Capitan are supported. It is typically necessary to ensure that the user running the software is currently logged into the desktop. No further user configuration should be necessary.

4 Firmware

4.1 Field Upgrades

4.1.1 Upgrade Procedure

FILL ME

5 API Documentation

5.1 Introduction

The PD analyzer API documentation that follows is oriented towards the PD analyzer Rosetta C bindings. The set of PD analyzer API functions and their functionality is identical regardless of which Rosetta language binding is utilized. The only differences will be found in the calling convention of the functions. For further information on such differences please refer to the documentation that accompanies each language bindings in the PD analyzer API Software distribution.

5.2 General Data Types

The following definitions are provided for convenience. All PD analyzer data types are unsigned.

```
typedef unsigned char       u08;
typedef unsigned short      u16;
typedef unsigned int        u32;
typedef unsigned long long  u64;
typedef signed   char       s08;
typedef signed   short      s16;
typedef signed   int        s32;
typedef signed   long long  s64;
typedef float               f32;
```

5.3 Notes on Status Codes

Most of the PD analyzer API functions can return a status or error code back to the caller. The complete list of status codes is provided at the end of this chapter and in the PD analyzer applications' user manuals. All of the error codes are assigned values less than 0, separating these responses from any numerical values returned by certain API functions.

5.4 General

5.4.1 Interface

Find Devices (pd_find_devices)

```
int pd_find_devices (int   num_devices,
                     u16 * devices);
```

Get a list of ports to which PD analyzers are attached.

Arguments

`num_devices`	maximum number of devices to return
`devices`	array into which the port numbers are returned

Return Value

This function returns the number of devices found, regardless of the array size.

Specific Error Codes

`None.`

Details

Each element of the array is written with the port number.

Devices that are in use are ORed with `PD_PORT_NOT_FREE` (`0x8000`). Under Linux, such devices correspond to PD analyzers that are currently in use. Under Windows, such devices are currently in use, but it is not known if the device is a PD analyzer.

Example:

Devices are attached to port 0, 1, 2
ports 0 and 2 are available, and port 1 is in-use.
`array` => { `0x0000, 0x8001, 0x0002` }

If the input array is `NULL`, it is not filled with any values.

If there are more devices than the array size (as specified by num_devices), only the first num_devices port numbers will be written into the array. Note that the devices array size in bytes must be at least twice the value specified in num_devices.

Find Devices (pd_find_devices_ext)

```
int pd_find_devices_ext (int   num_devices,
                         u16 * devices,
                         int   num_ids,
                         u32 * unique_ids);
```

Get a list of ports and unique IDs to which PD analyzers are attached.

Arguments

num_devices	maximum number of devices to return
devices	array into which the port numbers are returned
num_ids	maximum number of device IDs to return
unique_ids	array into which the unique IDs are returned

Return Value

This function returns the number of devices found, regardless of the array sizes.

Specific Error Codes

None.

Details

This function is the same as pd_find_devices() except that it also returns the unique IDs of each PD analyzer. The IDs are guaranteed to be non-zero if valid.

The IDs are the unsigned integer representation of the 10-digit serial numbers.

The number of devices and IDs returned in each of their respective arrays is determined by the minimum of num_devices and num_ids. However, if either array is NULL, the length passed in for the other array is used as-is, and the NULL array is not populated. If both arrays are NULL, neither array is populated, but the number of devices found is still returned.

Open a PD analyzer (pd_open)

```
PD pd_open (int port_number);
```

Open the PD port.

Arguments

`port_number`	The PD analyzer port number. This port number is the the same as the one obtained from the `pd_find_devices()` function. It is a zero-based number.

Return Value

This function returns a PD handle, which is guaranteed to be greater than zero if valid.

Specific Error Codes

`PD_UNABLE_TO_OPEN`	The specified port is not connected to a PD analyzer or the port is already in use.
`PD_INCOMPATIBLE_DEVICE`	There is a version mismatch between the DLL and the hardware. The DLL is not of a sufficient version for interoperability with the hardware version or vice versa. See `pd_open_ext()` in Section 5.4.1.4 for more information.

Details

This function is recommended for use in simple applications where extended information is not required. For more complex applications, the use of `pd_open_ext()` is recommended.

Open a PD analyzer (pd_open_ext)

```
PD pd_open_ext (int     port_number,
                PdExt * pd_ext);
```

Open the PD port, returning extended information in the supplied structure.

Arguments

`port_number` same as `pd_open`

`pd_ext` pointer to pre-allocated structure for extended version information available on open

Return Value

This function returns a PD handle, which is guaranteed to be greater than zero if valid.

Specific Error Codes

`PD_UNABLE_TO_OPEN` The specified port is not connected to a PD analyzer or the port is already in use.

`PD_INCOMPATIBLE_DEVICE` There is a version mismatch between the DLL and the hardware. The DLL is not of a sufficient version for interoperability with the hardware version or vice versa. The version information will be available in the memory pointed to by `pd_ext`.

Details

If NULL is passed as the pointer to the structure `pd_ext`, this function will behave exactly like `pd_open()`.

The `PdExt` structure is described below:

```
struct PdExt {
    PdVersion version;
    /* Feature bitmap for this device. */
    int features;
};
```

The `PdVersion` structure describes the various version dependencies of PD components. It can be used to determine which component caused an incompatibility error.

The features field denotes the capabilities of the PD analyzer. See the API function `pd_features` for more information.

```
struct PdVersion {
  /* Software, firmware, and hardware versions. */
  u16 software;
  u16 firmware;
  u16 hardware;

  /
* Firmware requires that software must be >= this version. */
  u16 sw_req_by_fw;

  /
* Software requires that firmware must be >= this version. */
  u16 fw_req_by_sw

  /
* Software requires that the API must be >= this version. */
  u16 api_req_by_sw;
};
```

All version numbers are of the format:

```
(major << 8) | minor
```

example: `v1.20` would be encoded as `0x0114`.

The structure is zeroed before the open is attempted. It is filled with whatever information is available. For example, if the hardware version is not filled, then the device could not be queried for its version number.

This function is recommended for use in complex applications where extended information is required. For simpler applications, the use of `pd_open()` is recommended.

Close a PD analyzer connection (pd_close)

```
int pd_close (Pd pd);
```

Close the PD analyzer port.

Arguments

pd	handle of a PD analyzer to be closed

Return Value

The number of analyzers closed is returned on success. This will usually be 1.

Specific Error Codes

```
None.
```

Details

If the `handle` argument is zero, the function will attempt to close all possible handles, thereby closing all open PD analyzer. The total number of PD analyzers closed is returned by the function.

Get Features (pd_features)

```
int pd_features (Pd pd);
```

Return the device features as a bit-mask of values, or an error code if the handle is not valid.

Arguments

`pd`	handle of a PD analyzer

Return Value

The features of the PD analyzer are returned. These are a bit-mask of the following values.

```
#define PD_FEATURE_NONE     (0)
#define PD_FEATURE_USBPD    (1<<0)
#define PD_FEATURE_USBPD_IV (1<<1)
```

Specific Error Codes

```
None.
```

Details

None.

Get Port (pd_port)

```
int pd_port (Pd pd);
```

Return the port number for this PD handle.

Arguments

`pd`	handle of a PD analyzer

Return Value

The port number corresponding to the given handle is returned. It is a zero-based number.

Specific Error Codes

```
None.
```

Details

None.

Get Unique ID (pd_unique_id)

```
u32 pd_unique_id (Pd pd);
```

Return the unique ID of the given PD analyzer.

Arguments

`pd`	handle of a PD analyzer

Return Value

This function returns the unique ID for this PD analyzer. The IDs are guaranteed to be non-zero if valid. The ID is the unsigned integer representation of the 10-digit serial number.

Specific Error Codes

```
None.
```

Details

None.

Status String (pd_status_string)

```
const char *pd_status_string (int status);
```

Return the status string for the given status code.

Arguments

`status`	status code returned by a PD API function

Return Value

This function returns a human readable string that corresponds to status. If the code is not valid, it returns a NULL.

Specific Error Codes

`None.`

Details

None.

Version (pd_version)

```
int pd_version (Pd pd, PdVersion *version);
```

Return the version matrix for the device attached to the given handle.

Arguments

`pd`	handle of a PD analyzer
`version`	pointer to pre-allocated structure

Return Value

A PD status code is returned with `PD_OK` on success.

Specific Error Codes

`PD_COMMUNICATION_ERROR`	The firmware of the specified device can not be determined.

Details

If the handle is 0 or invalid, only the software version is set.

See the details of `pd_open_ext()` for the definition of `PdVersion`.

Sleep (pd_sleep_ms)

```
u32 pd_sleep_ms (u32 milliseconds);
```

Sleep for given amount of time.

Arguments

`milliseconds`	number of milliseconds to sleep

Return Value

This function returns the number of milliseconds slept.

Specific Error Codes

`None.`

Details

This function provides a convenient cross-platform function to sleep the current thread using standard operating system functions.

The accuracy of this function depends on the operating system scheduler. This function will return the number of milliseconds that were actually slept.

5.4.2 Monitoring API

Start Capture (pd_capture_start)

```
int pd_capture_start (Pd pd);
```

Start monitoring packets.

Arguments

`pd`	handle of a PD analyzer

Return Value

A PD status code of `PD_OK` is returned on success.

Specific Error Codes

`None.`

Details

The blue LED on PD analyzer starts blinking when monitoring starts.

Stop Capture (pd_capture_stop)

```
int pd_capture_stop (Pd pd);
```

Stop monitoring packets.

Arguments

`pd`	handle of a PD analyzer

Return Value

A PD status code of `PD_OK` is returned on success.

Specific Error Codes

`None.`

Details

5.4.3 USB Power Delivery API

All read functions return a timestamp, a duration, a status, and an event value through the `PdReadInfo` parameter.

```
struct PdReadInfo {
    u64 timestamp;
    u64 duration;
    u32 status;
    u32 events;
  };
```

Table 1 : `PdReadInfo` structure

timestamp	timestamp when the packet or the event begins. This is the number of microseconds from where capture started and will be reset to 0 when `pa_capture_start` gets called.
duration	number of microseconds that the packet or the events actually took.
status	status bitmap as detailed in Tables 2.
events	events bitmap as detailed in Tables 3.

Table 2 : Read Status for Power Delivery

Valid with `pd_usbpd_read_bits` or `pd_usbpd_read_data` Error on packet decoding.		
`PD_STATUS_USBPD_ERR_MASK`	0x0000000F	Mask for the decoding error.
`PD_STATUS_USBPD_ERR_PREAMBLE`	0x00000001	Unable to decode or find preamble.
`PD_STATUS_USBPD_ERR_SOP`	0x00000002	Unable to decode or find SOP.
`PD_STATUS_USBPD_ERR_HEADER`	0x00000003	Unable to decode or find header/extended header.
`PD_STATUS_USBPD_ERR_DATA`	0x00000004	Unable to decode or find data.
`PD_STATUS_USBPD_ERR_CRC`	0x00000005	Unable to decode or find crc.
`PD_STATUS_USBPD_ERR_EOP`	0x00000006	Unable to decode or find EOP.
`PD_STATUS_USBPD_ERR_BAD_CRC`	0x01000000	Bad crc.
`PD_STATUS_USBPD_ERR_UNKNOWN_TYPE`	0x02000000	Unknown SOP type.

Table 3 : Read Status for Power Delivery

Valid with pd_usbpd_read_bits or pd_usbpd_read_data Packet information.		
PD_EVENT_USBPD_CC_MASK	0xF0000000	Mask for CC.
PD_EVENT_USBPD_CC_SHIFT	28	Number of shifts to get CC.
PD_EVENT_USBPD_CC1	0x00000000	CC1.
PD_EVENT_USBPD_CC2	0x10000000	CC2.
PD_EVENT_USBPD_POL_CHANGE	0x00000008	Detected polarity change on CC line.
PD_EVENT_USBPD_SOP_MASK	0x00000007	Mask for SOP type.
PD_EVENT_USBPD_SOP	0x00000000	SOP.
PD_EVENT_USBPD_SOP_PRIME	0x00000001	SOP'.
PD_EVENT_USBPD_SOP_DPRIME	0x00000002	SOP".
PD_EVENT_USBPD_SOP_PRIME_DEBUG	0x00000003	SOP' Debug.
PD_EVENT_USBPD_SOP_DPRIME_DEBUG	0x00000004	SOP" Debug.
PD_EVENT_USBPD_HARD_RESET	0x00000006	Hard reset.
PD_EVENT_USBPD_CABLE_RESET	0x00000007	Cable reset.
PD_EVENT_USBPD_EXTENDED_HEADER	0x00000010	Found extended header.
Valid with pd_usbpd_read_iv IV type.		
PD_EVENT_USBPD_IV_SOURCE_MASK	0x0F000000	Mask for IV source type.
PD_EVENT_USBPD_IV_SHIFT	24	Number of shifts to get CC.
PD_EVENT_USBPD_IV_VBUS_VOLTAGE	0x00000000	V_{BUS} Voltage.
PD_EVENT_USBPD_IV_VBUS_CURRENT	0x01000000	V_{BUS} Current.
PD_EVENT_USBPD_IV_VCONN_VOLTAGE	0x02000000	V_{CONN} Voltage.
PD_EVENT_USBPD_IV_VCONN_CURRENT	0x03000000	V_{CONN} Current.
PD_EVENT_USBPD_IV_CC1_VOLTAGE	0x04000000	CC1 Voltage. Measure upto 3.3V
PD_EVENT_USBPD_IV_CC2_VOLTAGE	0x06000000	CC2 Voltage. Measure upto 3.3V

Read USB Power Delivery Raw Packet (pd_usbpd_read_bits)

```
int pd_usbpd_read_bits (Pd           pd,
                        PdReadInfo * read_info,
                        u32        * bits_length,
                        u32        * preamble_length,
                                    u32          max_bytes,
                                    u08        * bits);
```

Read raw packet from CC line.

Arguments

`pd`	handle of a PD analyzer
`read_info`	filled with values described in Table 1
`bits_length`	number of bits of raw packet including all.
`preamble_length`	number of bits of preamble.
`max_bytes`	maximum number of data bytes to read
`bits`	an allocated array of `u08` which is filled with the received raw data

Return Value

This function returns the number of bytes read or a negative value indicating an error.

Specific Error Codes

`PD_READ_EMPTY`	No data to read.

Details

The USB PD Physical layer is encoded in Biphase Mark Coding(BMC). 0 is represented by a transition and 1 is represented by no transition in a single clock tick. Except for the preamble, all communications on the line is encoded in 4b5b code. `bits` conatins raw data after BMC decoding befor 4b5b decoding including preamble, SOP, data, checksum, and EOP.

`bits` is an array that contains raw data. The actual size of raw data can be any number of bits, so `bits_length` tells the exact number of bits of raw data. `preamble_length` is the number of bits of bit toggles before Start Of Packet.

In order to decode the 4b5b encoded data, please refer to `pd_usbpd_decode_bits`.

Decode USB Power Delivery Raw Packet (pd_usbpd_decode_bits)

```
int pd_usbpd_decode_bits (Pd          pd,
                                u32         bytes_length,
                                const u08 * bits,
                                u32       * header,
                                u32       * crc,
                                u32         max_bytes,
                                u08       * data);
```

Decode raw packet.

Arguments

`pd`	handle of a PD analyzer
`bytes_length`	number of bytes of `bits`
`bits`	raw packet received from `pd_usbpd_read_bits`.
`header`	header decoded from raw packet.
`crc`	crc decoded from raw packet.
`max_bytes`	maximum number of data bytes to read
`data`	an allocated array of `u08` which is filled with the decoded USB PD packet.

Return Value

This function returns the number of bytes decoded or a negative value indicating an error.

Specific Error Codes

`None.`

Details

This function decodes 4b5b encoded raw packet to USB PD packet. Since the number of bits of preamble and SOP type are already passed through `read_info` with `pd_usbpd_read_bits` function, this function decodes only from header to crc.

`header` & PD_HEADER_USBPD_STANDARD_MASK is the standard header and it always exists. ((`header` & PD_HEADER_USBPD_EXTENDED_MASK) >> 16) is the extended header only when PD_EVENT_USBPD_EXTENDED_HEADER is set in `read_info.events`.

data is the byte array containing data portion in the order it appears on the line. Since the way to parse the data portion varies for each command, please refer to the USB PD specification for more details.

Read USB Power Delivery Packet (pd_usbpd_read_bits)

```
int pd_usbpd_read_data (Pd            pd,
                        PdReadInfo * read_info,
                        u32        * preamble_length,
                        u32        * header,
                        u32        * crc,
                        u32          max_bytes,
                        u08        * data);
```

Read USB PD packet from CC line.

Arguments

pd	handle of a PD analyzer
read_info	filled with values described in Table 1
preamble_length	number of bits of preamble.
header	header decoded from raw packet.
crc	crc decoded from raw packet.
max_bytes	maximum number of data bytes to read
data	an allocated array of u08 which is filled with the decoded USB PD packet.

Return Value

This function returns the number of bytes decoded or a negative value indicating an error.

Specific Error Codes

PD_READ_EMPTY	No data to read.

Details

This function is combined of pd_usbpd_read_bits and pd_usbpd_decode_bits.

Read USB Power Delivery IV Data (pd_usbpd_read_bits)

```
int pd_usbpd_read_iv (Pd            pd,
                      PdReadInfo * read_info,
                      int        * value);
```

Read USB PD IV from CC line.

Arguments

`pd`	handle of a PD analyzer
`read_info`	filled with values described in Table 1
`value`	IV data.

Return Value

PD_OK for success.

Specific Error Codes

`PD_READ_EMPTY`	No data to read.
`PD_FUNCTION_NOT_AVAILABLE`	Firmware version doesn't support this feature.

Details

The USB Power Delivery Analyzer is rated for 5 A continuous and up to 20 V on V $_{BUS}$.

V_{BUS} Voltage - Measured via an INA231 ADC on the V_{BUS} lines of the USB Type C receptacle connector pins A4, A9, B4, and B9.

V_{BUS} Current - Measured via an INA231 ADC across a 0.015 Ohm shunt resistor between the V_{BUS} input Type C receptacle and the V_{BUS} Type C plug.

V_{CONN} Voltage - Measured on CC2 signal (pin B5) of the USB Type C receptacle - Please note you may need to flip the PDA over if V_{CONN} ends up on the CC1 signal pin (A5).

V_{CONN} Current - Measured across a 0.015 Ohm shunt resistor between the CC2 (pin B5) Type C receptacle and the CC2 (pin B5) Type C plug.

CC1/CC2 Voltage - Up to 3.3V and is useful as "traffic indication".

Disclaimer: When using the USB Power Delivery Analyzer above the rated current and voltage, extreme caution is advised. Customers who choose to do so are at their own risk and may cause permanent damage to the analyzer. Total Phase is not liable for damages caused by applying current and voltage in excess of the warranted operating range.

A measurement is taken on each of the 6 channels, once approximately every 8 milliseconds, in a sequential (round-robin) fashion. The previous channel values are held in the graph until the next valid measurement.

The type of IV data can be retrieved from `read_info.events` and if it is for current, it can be negative which represents the direction of power.

This function returns `PD_FUNCTION_NOT_AVAILABLE` with any firmware below v1.10. Please make sure the firmware version is equal to or above v1.10.

5.5 Error Codes

Table 4 : PD API Error Codes

Literal Name	**Value**	**pd_status_string() return value**
PD_OK	0	ok
PD_UNABLE_TO_LOAD_LIBRARY	-1	unable to load library
PD_UNABLE_TO_LOAD_DRIVER	-2	unable to load usb driver
PD_UNABLE_TO_LOAD_FUNCTION	-3	unable to load function
PD_INCOMPATIBLE_LIBRARY	-4	incompatible library version
PD_INCOMPATIBLE_DEVICE	-5	incompatible device version
PD_INCOMPATIBLE_DRIVER	-6	incompatible device version
PD_COMMUNICATION_ERROR	-7	communication error
PD_UNABLE_TO_OPEN	-8	unable to open device
PD_UNABLE_TO_CLOSE	-9	unable to close device
PD_INVALID_HANDLE	-10	invalid device handle

PD_CONFIG_ERROR	-11	configuration error
PD_STILL_ACTIVE	-12	device still active
PD_FUNCTION_NOT_AVAILABLE	-13	function not available
PD_READ_EMPTY	-100	nothing to read

6 Legal / Contact

6.1 Disclaimer

All of the software and documentation provided in this manual, is copyright Total Phase, Inc. ("Total Phase"). License is granted to the user to freely use and distribute the software and documentation in complete and unaltered form, provided that the purpose is to use or evaluate Total Phase products. Distribution rights do not include public posting or mirroring on Internet websites. Only a link to the Total Phase download area can be provided on such public websites.

Total Phase shall in no event be liable to any party for direct, indirect, special, general, incidental, or consequential damages arising from the use of its site, the software or documentation downloaded from its site, or any derivative works thereof, even if Total Phase or distributors have been advised of the possibility of such damage. The software, its documentation, and any derivative works is provided on an "as-is" basis, and thus comes with absolutely no warranty, either express or implied. This disclaimer includes, but is not limited to, implied warranties of merchantability, fitness for any particular purpose, and non-infringement. Total Phase and distributors have no obligation to provide maintenance, support, or updates.

Information in this document is subject to change without notice and should not be construed as a commitment by Total Phase. While the information contained herein is believed to be accurate, Total Phase assumes no responsibility for any errors and/or omissions that may appear in this document.

6.2 Life Support Equipment Policy

Total Phase products are not authorized for use in life support devices or systems. Life support devices or systems include, but are not limited to, surgical implants, medical systems, and other safety-critical systems in which failure of a Total Phase product could cause personal injury or loss of life. Should a Total Phase product be used in such an unauthorized manner, Buyer agrees to indemnify and hold harmless Total Phase, its officers, employees, affiliates, and distributors from any and all claims arising from such use, even if such claim alleges that Total Phase was negligent in the design or manufacture of its product.

6.3 Contact Information

Total Phase can be found on the Internet at http://www.totalphase.com/. If you have support-related questions, please go to the Total Phase support page at http://www.totalphase.com/support/. For sales inquiries, please contact sales@totalphase.com .