
# USB Power Delivery Exerciser Manual

Manual Version 2.34

**For USB Protocol Suite v9.71 and above**

**February 2025**

**WARNING:** Information contained herein is classified as EAR99 under the U.S. Export Administration Regulations. Export, reexport or diversion contrary to U.S. law is prohibited.

## Document Disclaimer

The information contained in this document has been carefully checked and is believed to be reliable. However, no responsibility can be assumed for inaccuracies that may not have been detected.

Teledyne LeCroy reserves the right to revise the information presented in this document without notice or penalty.

## Trademarks and Servicemarks

*CATC Trace, Voyager M310C, Voyager M310P, Voyager M310e, Voyager M4x, Voyager ReadyLink, USB Protocol Suite,* and *BusEngine* are trademarks of Teledyne LeCroy.

All other trademarks are property of their respective companies.

## Copyright

Copyright © 2020, Teledyne LeCroy, Inc. All Rights Reserved.

This document may be printed and reproduced without additional permission, but all copies should contain this copyright notice.

# Contents



**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

# 1 Introduction

Integrated in Teledyne LeCroy's Voyager M310C test platform, the Power Delivery exerciser supports traffic generation, including both provider and consumer device emulation. The Power Delivery exerciser continues to evolve with each software release. Be sure to check for updated software and firmware before getting started with the Exerciser.

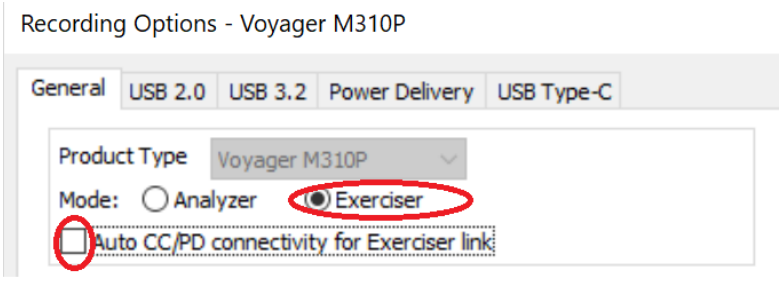
## 1.1 Scope of the Document

The document is intended as an extension to Voyager Exerciser language. It addresses only the new elements specific to PD Exerciser on top of the language described in "**Voyager Exerciser Language Manual**" (accessible through Menu -> Help -> Other Manuals).

## 1.2 Getting Started


- The port that labeled as "Exerciser" should be used to connect DUT to the PD Exerciser. This port is same as left Analyzer port in Voyager M310C, but in other platforms "Exerciser" port is independent port.
- The PD exerciser also requires specific cable orientation (Red LED when connected wrong side-up). Voyager M310P platform is capable to work in every orientation.

- To enable the PD Trainer/Exerciser, set working mode to "Exerciser" in general page of "Recording options" and make sure "Auto CC/PD connectivity" is unchecked.

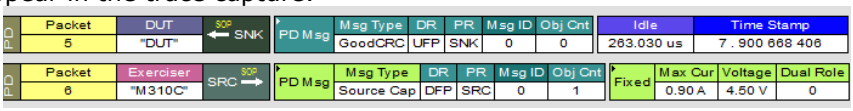



- If need to source more than 5V need to uncheck "Allow Vbus > 5V" in "Power delivery tab of "Recording Options". See important note below:


**Note** – *Allow VBUS > 5v* is a safety feature which prevents sourcing above 5V. When enabled, this mode will allow Voltage levels to be delivered to the DUT which may exceed their current carrying capabilities. While the Voyager system is designed to tolerate higher current, these higher voltages may inadvertently cause damage to devices/cables under test.

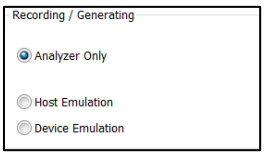
- To set *devices port name*, use the General Tab under "*Recording Options"* to add "alias labels" for your DUTs.

These labels will appear in the trace capture.




The Alias name is primarily for use in analyzer mode and requires that device names are added before recording traffic. The device naming can also be used in Exerciser mode; however message frames from the Voyager M310C will be always be labeled "M310C".

- Within the USB 3.1 tab – "*Recording/Generating"* option - 'Analyzer Only' mode unless you also want to run 3.1




leave in traffic.

- Use the example PD Exerciser scripts to begin testing:

C:\Users\Public\Documents\LeCroy\USB Protocol Suite\Examples\Power Delivery Exerciser

| Example Script | Behavior |
|---|---|
| **Source Power Negotiate VDM.updg** | Voyager as Source negotiates default Provider 900mA@4.5V then sends Discover-Id. Using Basic Commands. |
| **High Level Negotiate with dynamic change cap.updg** | Voyager as Source negotiates default Provider 1A@5V then broadcasts lower PDO 900mA@4.5V and re-negotiates. Using High Level Commands. |
| **Discover Cable.updg** | Voyager as Source programmatically turns on VCONN and performs Discovery Process for cable. Using High Level Commands. |
| **Sink Power Negotiate.updg** | Voyager as Sink Waits to receive Source cap then negotiates as Sink - 900mA@5V. Using Basic Commands. |
| **Apple VGA multiple Adaptor.updg** | Voyager as Source enables VCONN and Sends Discover Id; Discover Mode for Apple SVID (0x05AC); Enter Mode (PD_DISPLAY_PORT_SVID) then Exit Mode; turns off VCONN. Using Basic Commands. |
| **High Level Device Discovery.updg** | Voyager as Source sends Discover Id; Discover SVIDs; Discover Modes for Display Port SVID (0xFF01); Enter Mode (0xFF01); Exit Mode (0xFF01); Discover Modes for Apple SVID (0x05AC); Enter Mode(0x5AC mode 1); Exit Mode(0x5AC mode 1); Enter Mode(0x5AC mode 2); Exit Mode(0x5AC mode 2); Using High Level Commands. |
| **NegotiationSample_WithSwapPowerRole.updg** | Voyager as Source sends SwapPowerRole; and negotiates as a Sink after power role swap. 1.5A@5V. Using High Level Commands. |
| **Sink Auto Response.updg** | Voyager as Sink will response to all incoming PD messages within 100s. Using Auto Response Command. |

- To Run Sample Script – Connect Cable to Exerciser port; Click *Record*, wait a few seconds and Click *Run.* The PD Exerciser uses the sequence below at the beginning of each example script to simulate a re-connect event.

```
call PD_Disconnect()

call PD_SetResistorRp( PD_ON, CC_RP_CUR_1_5, CC_LINE_1 )
call PD_SetVBus( PD_ON )
```

**Note**- it's also possible to execute the example scripts before the cable is connected to M310C then performing "hot-plug" (It's possible some issues may be seen with some devices not responding to exerciser in this case).
**Note** – some latency may be observed when activating/downloading PD exerciser scripts (Run button) This will be improved in a future release.

**Important Licensing Note:**

- Operating the PD Exerciser beta requires that the USB Power Delivery Exerciser option is enabled on the M310C base unit:

| | | |
|---|---|---|
| USB Power Delivery - Type C | Yes | USB Power Delivery Analysis - Type-C |
| USB Power Delivery - Exerciser | Yes | USB Power Delivery Exerciser |

## 1.3 Power Delivery with EPR

In this document, exerciser commands and modes designated for PD with EPR (as both PD Source and PD Sink) are only supported on the Voyager M310e platform. Using these commands/modes with other platforms may lead to compilation errors and scripts that do not complete.

In PD Source mode, if capacity in the extended range (EPR) is required (voltage > 20V), a compatible programmable power supply must be connected through EPR power/control interface on the rear of the Voyager M310e.


*Figure 1: EPR Power / Control connectors*

The following power supplies are currently supported:

- B&K Precision 9202/B
- B&K Precision 9205/B
- B&K Precision 9206/B

The USB Protocol Suite software will integrate seamlessly with the power supply, so there's no need for any additional programming. Simply define the PD Source capabilities using the commands found in this document, and everything will be configured automatically behind the scenes.

# 2 Packet Templates

Following Packet Templates can be used in Basic or High-Level commands as data containers. All of these messages inherited from `PD_Packet` packet template except those which are used as containers for Data Objects.

## 2.1 PD_MessageHeader

### 2.1.1 Revision 2.0

```
Packet PD_MessageHeader : PD_Packet
{
   MessageType                : 4 = 0
   Reserved1                  : 1 = 0
   PortDataRole_Reserved2     : 1 = 0
   SpecificationRevision      : 2 = 1
   PortPowerRole_CablePlug    : 1 = 0
   MessageId                  : 3 = 0
   NumberOfDataObjects        : 3 = 0
   Reserved2                  : 1 = 0
}
```

### 2.1.2 Revision 3.0

```
Packet PD_MessageHeader : PD_Packet
{
   MessageType                : 5 = 0
   PortDataRole_Reserved2     : 1 = 0
   SpecificationRevision      : 2 = 2
   PortPowerRole_CablePlug    : 1 = 0
   MessageId                  : 3 = 0
   NumberOfDataObjects        : 3 = 0
   Extended                   : 1 = 0
}
```

## 2.2 PD_ControlMessage

```
Packet PD_ControlMessage : PD_Packet
{
   : PD_MessageHeader
}
```

## 2.3 PD_GoodCrcMessage

```
Packet PD_GoodCrcMessage : PD_ControlMessage
```

```
{
  MessageType    = PD_MESSAGE_TYPE_GOODCRC
}
```

## 2.4 PD_GotoMinMessage

```
Packet PD_GotoMinMessage : PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_GOTO_MIN
}
```

## 2.5 PD_AcceptMessage

```
Packet PD_AcceptMessage : PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_ACCEPT
}
```

## 2.6 PD_RejectMessage

```
Packet PD_RejectMessage : PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_REJECT
}
```

## 2.7 PD_PingMessage

```
Packet PD_PingMessage: PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_PING
}
```

## 2.8 PD_PsRdyMessage

```
Packet PD_PsRdyMessage: PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_PS_RDY
}
```

## 2.9 PD_GetSourceCapMessage

```
Packet PD_GetSourceCapMessage: PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_GET_SOURCE_CAP
}
```

## 2.10 PD_GetSinkCapMessage

```
Packet PD_GetSinkCapMessage: PD_ControlMessage
{
  MessageType    = PD_MESSAGE_TYPE_GET_SINK_CAP
}
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 2.11 PD_DataRoleSwapMessage

```
Packet PD_DataRoleSwapMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_DR_SWAP
}
```

## 2.12 PD_PowerRoleSwapMessage

```
Packet PD_PowerRoleSwapMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_PR_SWAP
}
```

## 2.13 PD_VConnSwapMessage

```
Packet PD_VconnSwapMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_VCONN_SWAP
}
```

## 2.14 PD_WaitMessage

```
Packet PD_WaitMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_WAIT
}
```

## 2.15 PD_SoftResetMessage

```
Packet PD_SoftResetMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_SOFT_RESET
}
```

## 2.16 PD_DataResetMessage

```
Packet PD_DataResetMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_DATA_RESET
}
```

## 2.17 PD_DataResetCompleteMessage

```
Packet PD_DataResetCompleteMessage: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_DATA_RESET_COMPLETE
}
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 2.18 PD_NotSupportedMsg

```
Packet PD_NotSupportedMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_NOT_SUPPORTED
}
```

## 2.19 PD_GetSourceCapExtendedMsg

```
Packet PD_GetSourceCapExtendedMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_GET_SRC_CAP_EXT
}
```

## 2.20 PD_GetStatusMsg

```
Packet PD_GetStatusMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_GET_STATUS
}
```

## 2.21 PD_FRSwapMsg

```
Packet PD_FRSwapMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_FR_SWAP
}
```

## 2.22 PD_GetPPSStatusMsg

```
Packet PD_GetPPSStatusMsg: PD_ControlMessage
{
   MessageType    =PD_MESSAGE_TYPE_GET_PPS_STATUS
}
```

## 2.23 PD_GetCountryCodesMsg

```
Packet PD_GetCountryCodesMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_GET_COUNTRY_CODES
}
```

## 2.24 PD_GetSinkCapExtendedMsg

```
Packet PD_GetSinkCapExtendedMsg: PD_ControlMessage
{
   MessageType    = PD_MESSAGE_TYPE_GET_SNK_CAP_EXT
}
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 2.25 PD_SourceCapabilitiesMessage

```
Packet PD_SourceCapabilitiesMessage : PD_ControlMessage
{
   MessageType                = 1
   PortPowerRole_CablePlug    = 1
   NumberOfDataObjects        = 0

   SourceCapabilitiesData     : *   This field can contain one or more (up-to 7 according to the PD Spec)
                                    PDOs. Available PDO templates are:
                                    PD_PowerDataObjectFixedSupply_Source,
                                    PD_PDOFixedSupplyNotVSafe5V_Source,
                                    PD_PowerDataObjectVariableSupply_Source,
                                    PD_PowerDataObjectBatterySupply_Source,
                                    PD_PowerDataObjectPPS_Source,
                                    PD_SPRPowerDataObjectAVS
}
```

### 2.25.1 PD_PowerDataObjectFixedSupply_Source

```
Packet PD_PowerDataObjectFixedSupply_Source : PD_PowerDataObject
{
   MaxCurrent_10mAUnits          : 10 = 100
   Voltage_50mVUnits             : 10 = 100
   PeakCurrent                   : 2  = 0
   Reserved                      : 1  = 0
   EPRModeCapable                : 1  = 0
   UnchunkedExtMsgSupported      : 1  = 1
   DualRoleData                  : 1  = 0
   UsbCommunicationsCapable      : 1  = 0
   UnconstrainedPower            : 1  = 1
   UsbSuspendSupported           : 1  = 0
   DualRolePower                 : 1  = 0
   PowerDataType                 : 2  = 0
}
```

### 2.25.2 PD_PDOFixedSupplyNotVSafe5V_Source

```
Packet PD_PDOFixedSupplyNotVSafe5V_Source : PD_PowerDataObject
{
   MaxCurrent_10mAUnits    : 10 = 0
   Voltage_50mVUnits       : 10 = 0
   PeakCurrent             : 2  = 0
   Reserved                : 8  = 0
   PowerDataType           : 2  = 0
}
```

### 2.25.3 PD_PowerDataObjectVariableSupply_Source

```
Packet PD_PowerDataObjectVariableSupply_Source : PD_PowerDataObject
{
   MaxCurrent_10mAUnits     : 10 = 0
   MinVoltage_50mVUnits     : 10 = 0
   MaxVoltage_50mVUnits     : 10 = 0
   PowerDataType            : 2  = 2
}
```

### 2.25.4 PD_PowerDataObjectBatterySupply_Source

```
Packet PD_PowerDataObjectBatterySupply_Source : PD_PowerDataObject
{
   MaxAllowablePower_250mWUnits   : 10 = 0
   MinVoltage_50mVUnits           : 10 = 0
   MaxVoltage_50mVUnits           : 10 = 0
   PowerDataType                  : 2  = 1
}
```

### 2.25.5 PD_PowerDataObjectPPS_Source

```
Packet PD_PowerDataObjectPPS_Source : PD_PowerDataObject
{
   MaxCurrent_50mAUnits     : 7 = 0
   PPS_Rsvd_1               : 1 = 0
   MinVoltage_100mVUnits    : 8 = 0
   PPS_Rsvd_2               : 1 = 0
   MaxVoltage_100mVUnits    : 8 = 0
   PPS_Rsvd_3               : 2 = 0
   PPSPowerLimited          : 1 = 0
   APDOType                 : 2 = 0
   PowerDataType            : 2 = 3
}
```

### 2.25.6 PD_SPRPowerDataObjectAVS

```
Packet PD_SPRPowerDataObjectAVS : PD_PowerDataObject
{
   MaxCurrent9To15VRange_10mAUnits      : 10 = 0
   MaxCurrent15To20VRange_10mAUnits     : 10 = 0
   SPRAVS_Rsvd_1                        : 6 = 0
   PeakCurrent                          : 2 = 0
   APDOType                             : 2 = 2
   PowerDataType                        : 2 = 3
}
```

## 2.26 PD_SinkCapabilitiesMessage

```
Packet PD_SinkCapabilitiesMessage : PD_ControlMessage
{
```

```
    MessageType                 = 4
    PortPowerRole_CablePlug     = 0
    NumberOfDataObjects         = 1

    SinkCapabilitiesData        : *     This field can contain one or more (up-to 7 according to the PD Spec)
                                        PDOs. Available PDO templates are:
                                        PD_PowerDataObjectFixedSupply_Sink,
                                        PD_PowerDataObjectVariableSupply_Sink,
                                        PD_PowerDataObjectBatterySupply_Sink,
                                        PD_PowerDataObjectPPS_Sink,
                                        PD_SPRPowerDataObjectAVS
}
```

### 2.26.1 PD_PowerDataObjectFixedSupply_Sink

```
Packet PD_PowerDataObjectFixedSupply_Sink : PD_PowerDataObject
{
    OperationalCurrent_10mAUnits    : 10 = 100
    Voltage_50mVUnits               : 10 = 100
    Reserved                        : 3  = 0
    FRSwapTypeCCurrent              : 2  = 0
    DualRoleData                    : 1  = 0
    UsbCommunicationsCapable        : 1  = 0
    UnconstrainedPower              : 1  = 1
    HigherCapability                : 1  = 0
    DualRolePower                   : 1  = 0
    PowerDataType                   : 2  = 0
}
```

### 2.26.2 PD_PDOFixedSupplyNotVSafe5V_Sink

```
Packet PD_ PDOFixedSupplyNotVSafe5V_Sink : PD_PowerDataObject
{
    OperationalCurrent_10mAUnits    : 10 = 100
    Voltage_50mVUnits               : 10 = 100
    Reserved                        : 10  = 0
    PowerDataType                   : 2  = 0
}
```

### 2.26.3 PD_PowerDataObjectVariableSupply_Sink

```
Packet PD_PowerDataObjectVariableSupply_Sink : PD_PowerDataObject
{
    OperationalCurrent_10mAUnits    : 10 = 0
    MinVoltage_50mVUnits            : 10 = 0
    MaxVoltage_50mVUnits            : 10 = 0
    PowerDataType                   : 2  = 2
}
```

### 2.26.4 PD_PowerDataObjectBatterySupply_Sink

```
Packet PD_PowerDataObjectBatterySupply_Sink : PD_PowerDataObject
{
   OperationalPower_250mWUnits    : 10 = 0
   MinVoltage_50mVUnits           : 10 = 0
   MaxVoltage_50mVUnits           : 10 = 0
   PowerDataType                  : 2  = 1
}
```

### 2.26.5 PD_PowerDataObjectPPS_Sink

```
Packet PD_PowerDataObjectPPS_Sink : PD_PowerDataObject
{
   MaxCurrent_50mAUnits     : 7 = 0
   PPS_Rsvd_1               : 1 = 0
   MinVoltage_100mVUnits    : 8 = 0
   PPS_Rsvd_2               : 1 = 0
   MaxVoltage_100mVUnits    : 8 = 0
   PPS_Rsvd_3               : 3 = 0
   APDOType                 : 2 = 0
   PowerDataType            : 2 = 3
}
```

### 2.26.6 PD_SPRPowerDataObjectAVS

Refer to `PD_SPRPowerDataObjectAVS`.

## 2.27 PD_RequestPacket

```
Packet PD_RequestPacket : PD_MessageHeader
{
   MessageType              = 2
   NumberOfDataObjects      = 1

   Data                     : 32 = 0x00
}
```

This field can contain only one RDO packet variables. Available RDO templates are:
`PD_RequestDataObject_Fixed_Variable_NoGiveBack`,
`PD_RequestDataObject_Fixed_Variable_GiveBack`,
`PD_RequestDataObject_Battery_NoGiveBack`,
`PD_RequestDataObject_Battery_GiveBack`,
`PD_ProgrammableRDO`

### 2.27.1 PD_RequestDataObjectCommon

```
Packet PD_RequestDataObjectCommon
{
   Rsvd1                    : 2 = 0
   EPRModeCapable           : 1 = 0
```

```
  UnchunkedExtMsgSupported    : 1 = 1
  NoUsbSuspend                : 1 = 0
  UsbCommunicationsCapable    : 1 = 0
  CapabilityMismatch          : 1 = 0
  GiveBackFlag                : 1 = 0
  ObjectPosition              : 4 = 1
}
```

### 2.27.2 PD_RequestDataObject_Fixed_Variable_NoGiveBack

```
Packet PD_RequestDataObject_Fixed_Variable_NoGiveBack : PD_RequestData
{
  MaxOperatingCurrent_10mAUnits    : 10 = 0
  OperatingCurrent_10mAUnits       : 10 = 0
  : PD_RequestDataObjectCommon
}
```

### 2.27.3 PD_RequestDataObject_Fixed_Variable_GiveBack

```
Packet PD_RequestDataObject_Fixed_Variable_GiveBack : PD_RequestData
{
  MinOperatingCurrent_10mAUnits    : 10 = 0
  OperatingCurrent_10mAUnits       : 10 = 0

  : PD_RequestDataObjectCommon

  GiveBackFlag                     = 1
}
```

### 2.27.4 PD_RequestDataObject_Battery_NoGiveBack

```
Packet PD_RequestDataObject_Battery_NoGiveBack : PD_RequestData
{
  MaxOperatingPower_250mWUnits    : 10 = 0
  OperatingPower_250mWUnits       : 10 = 0

  : PD_RequestDataObjectCommon
}
```

### 2.27.5 PD_RequestDataObject_Battery_GiveBack

```
Packet PD_RequestDataObject_Battery_GiveBack : PD_RequestData
{
  MinOperatingPower_250mWUnits    : 10 = 0
  OperatingPower_250mWUnits       : 10 = 0

  : PD_RequestDataObjectCommon

  GiveBackFlag                    = 1
```

```
}
```

### 2.27.6 PD_ProgrammableRDO

```
Packet PD_ProgrammableRDO : PD_RequestData
{
   OperatingCurrent_50mAUnits        : 7  = 0
   PRDO_Rsvd_1                       : 2  = 0
   OutputVoltage_20mVUnits           : 11 = 0

   : PD_RequestDataObjectCommon
}
```

## 2.28 PD_BISTMessage

```
Packet PD_BISTMessage : PD_ControlMessage
{
   MessageType              = 3
   NumberOfDataObjects      = 1

   : PD_BISTDataObject

   TestData                 : *
}
```

### 2.28.1 PD_BISTDataObject

```
Packet PD_BISTDataObject
{
   Reserved             : 28 = 0x00
   BISTRequestType      : 4  = BIST_REQUEST_CARRIER_MODE
}
```

## 2.29 PD_BISTCarrierModeMessage

```
Packet PD_BISTCarrierModeMessage : PD_BISTMessage
{
}
```

## 2.30 PD_BISTTestDataMessage

```
Packet PD_BISTTestDataMessage : PD_BISTMessage
{
   NumberOfDataObjects      = 7
   BISTRequestType          = BIST_REQUEST_TEST_DATA
```

```
  TestData                = { AA AA AA AA
                              AA AA AA AA
                              AA AA AA AA
                              AA AA AA AA
                              AA AA AA AA
                              AA AA AA AA }      Maximum 192 bytes
                                                 Should be multiple of 32b
}
```

## 2.31 PD_BISTSharedCapacityTestModeEntry

```
Packet PD_BISTSharedTestModeEntry : PD_BISTMessage
{
  BISTRequestType    = BIST_REQUEST_SHARED_TEST_MODE_ENTRY
}
```

## 2.32 PD_BISTSharedCapacityTestModeExit

```
Packet PD_BISTSharedTestModeExit : PD_BISTMessage
{
  BISTRequestType    = BIST_REQUEST_SHARED_TEST_MODE_EXIT
}
```

## 2.33 PD_BatteryStatusMsg

```
Packet PD_BatteryStatusMsg : PD_ControlMessage
{
  MessageType                = 5
  NumberOfDataObjects        = 1

  : PD_BatteryStatusDataObject
}
```

### 2.33.1 PD_BatteryStatusDataObject

```
Packet PD_BatteryStatusDataObject
{
  Reserved_1                 : 8  = 0x00
  InvalidBatteryReference    : 1  = 0x00
  BatteryIsPresent           : 1  = 0x00
  BatteryChargingStatus      : 2  = 0x00
  Reserved_2                 : 4  = 0x00
  BatteryPC_100mWHUnits      : 16 = 0xFFFF
}
```

## 2.34 PD_AlertMsg

```
Packet PD_AlertMsg : PD_ControlMessage
{
```

```
   MessageType              = 6
   NumberOfDataObjects      = 1

   : PD_AlertDataObject
}
```

### 2.34.1 PD_AlertDataObject

```
Packet PD_AlertDataObject
{
   ExtendedAlertEventType      : 4  = 0x00
   Reserved_1                  : 12 = 0x00
   HotSwappableBatteries       : 4  = 0x00
   FixedBatteries              : 4  = 0x00
   Reserved_2                  : 1  = 0x00
   BatteryStatusChange         : 1  = 0x00
   OverCurProtection           : 1  = 0x00
   OverTempProtection          : 1  = 0x00
   OperatingConditionChange    : 1  = 0x00
   SourceInputChange           : 1  = 0x00
   OverVoltageProtection       : 1  = 0x00
   ExtendedAlertEvent          : 1  = 0x00
}
```

## 2.35 PD_GetCountryInfoMsg

```
Packet PD_GetCountryInfoMsg : PD_ControlMessage
{
   MessageType              = 7
   NumberOfDataObjects      = 1

   : PD_GetCountryInfoDO
}
```

### 2.35.1 PD_GetCountryInfoDO

```
Packet PD_GetCountryInfoDO
{
   GetCountryInfoDO_Rsvd_1     : 16 = 0x00
   Alpha2CountryCode2ndChar    : 8 = 0x00
   Alpha2CountryCode1stChar    : 8 = 0x00
}
```

## 2.36 PD_EnterUSB_Message

```
Packet PD_EnterUSB_Message : PD_ControlMessage
{
   MessageType                = 8
   NumberOfDataObjects        = 1

   : PD_EnterUSBDataObject
}
```

### 2.36.1 PD_EnterUSBDataObject

```
Packet PD_EnterUSBDataObject
{
   Reserved_1     : 13 = 0x00
   HostPresent    : 1  = PD_FALSE
   TBTSupport     : 1  = PD_FALSE
   DPSupport      : 1  = PD_FALSE
   PCIeSupport    : 1  = PD_FALSE
   CableCurrent   : 2  = PD_CABLECURRENT_VBUS_NOT_SUPP
   CableType      : 2  = PD_CABLETYPE_PASSIVE
   CableSpeed     : 3  = PD_CABLESPEED_USB2_NO_SSP
   Reserved_2     : 1  = 0x00
   USB3DRD        : 1  = PD_FALSE
   USB4DRD        : 1  = PD_FALSE
   Reserved_3     : 1  = 0x00
   USBMode        : 3  = PD_USBMODE_USB2
   Reserved_4     : 1  = 0x00
}
```

## 2.37 PD_VDM_Unstructured_Header

```
Packet PD_VDM_Unstructured_Header : PD_ControlMessage
{
   MessageType            = 15
   NumberOfDataObjects    = 0x01

   VDMCustom              : 15 = 0x00
   VDMType                : 1  = PD_VDM_TYPE_UNSTRUCTURED_VDM
   VDMSVID                : 16 = 0x00
}
```

The following is an example which demonstrates how to send an Unstructured VDM containing two VDOs:

```
# Define VDO1 fields
packet VDO1Type
{
    Field1 : 2  = 0x03
    Field2 : 10 = 0xfff
    Field3 : 20 = 0xffffff
}
local $vdo1_var = VDO1Type
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
# Create an unstructured VDM packet with two VDOs
local $unstructured_vdm = PD_VDM_Unstructured_Header
{
    # Number of VDOs + 1
    NumberOfDataObjects = 0x03

    # Vendor SVID
    VDMSVID = 0xABCD

    # Optional Packet variable
    VDO1 : 32 = $vdo1_var

    # Optional value
    VDO2 : 32 = 0x11111111
}

# Send VDM packet with default settings
local $SendSettings = PD_SendPacketSettings
call PD_SendPacket($unstructured_vdm, $SendSettings)

# To receive probable response packet
local $ReceiveSettings = PD_ReceivePacketSettings
call PD_ReceivePacket($ReceiveSettings)
```

## 2.38 PD_VDM_Structured_Header

```
Packet PD_VDM_Structured_Header : PD_ControlMessage
{
  MessageType                 = 15
  NumberOfDataObjects         = 0x01

  VDMCommand                  : 5  = PD_VDM_COMMAND_RESERVED_0
  VDMReserved1                : 1  = 0x00
  VDMCommandType              : 2  = PD_VDM_COMMAND_TYPE_REQ
  VDMObjectPosition           : 3  = 0x00
  VDMReserved2                : 2  = 0x00
  VDMStructuredVdmVersion     : 2  = PD_VDM_STRUCTURED_VERSION_2
  VDMType                     : 1  = PD_VDM_TYPE_STRUCTURED_VDM
  VDMSVID                     : 16 = 0x00
}
```

## 2.39 PD_VDM_Discover_Identity_Message

```
Packet PD_VDM_Discover_Identity_Message : PD_VDM_Structured_Header
{
  VDMCommand    = PD_VDM_COMMAND_DISCOVER_IDENTITY
  VDMSVID       = PD_VDM_SID
}
```

## 2.40 PD_VDM_Discover_Identity_Response

```
Packet PD_VDM_Discover_Identity_Response : PD_VDM_Discover_Identity_Message
{
  VDMCommandType    = PD_VDM_COMMAND_TYPE_RESPONDER_ACK

  VDOs              : *     #This field can contain up-to 6 VDOs, but should contain at least 3
                            VDOs (according to PD Spec). Available VDO themplates are:
                            PD_VDM_Discover_Identity_ID_Header_VDO,
                            PD_VDM_Discover_Identity_Cert_Stat_VDO,
                            PD_VDM_Discover_Identity_Product_VDO,
                            PD_VDM_Discover_Identity_Cable_VDO (Rev 2.0 only),
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
                                    PD_DiscoverId_DFP_VDO (Rev 3.0 only),
                                    PD_DiscoverId_UFP_1_VDO (Rev 3.0 only),
                                    PD_DiscoverId_UFP_2_VDO (Rev 3.0 only),
                                    PD_DiscoverIdPassiveCableVdo (Rev 3.0 only),
                                    PD_DiscoverIdActiveCableVdo_1 (Rev 3.0 only),
                                    PD_DiscoverIdActiveCableVdo_2 (Rev 3.0 only),
                                    PD_VDM_Discover_Identity_Alternate_Mode_Adapter_VDO,
                                    PD_DiscoverIdVConnPoweredDeviceVdo (Rev 3.0 only)
}
```

### 2.40.1 PD_VDM_Discover_Identity_ID_Header_VDO

#### 2.40.1.1 Revision 2.0

```
Packet PD_VDM_Discover_Identity_ID_Header_VDO : PD_DiscoverIdentity_VDO
{
   IDHeaderVDO_USBVendorID                 : 16 = 0x05FF #LeCroy ID
   IDHeaderVDO_Reserved                    : 10 = 0x00
   IDHeaderVDO_ModalOperationSupported     : 1  = 0x00
   IDHeaderVDO_ProductType                 : 3  =
                                             PD_VDM_ID_HEADER_VDO_PRODUCT_TYPE_U
                                             NDEFINED
   IDHeaderVDO_DataCapableAsUSBDevice      : 1  = 0x00
   IDHeaderVDO_DataCapableAsUSBHost        : 1  = 0x00
}
```

#### 2.40.1.2 Revision 3.0

```
Packet PD_VDM_Discover_Identity_ID_Header_VDO : PD_DiscoverIdentity_VDO
{
   USBVendorID                : 16 = 0x05FF #LeCroy ID
   Reserved                   : 5  = 0x00
   ConnectorType              : 2  = PD_CABLE_CONNECTOR_TYPE_TYPEC_RECEPTACLES
   ProductType_DFP            : 3  = PD_PRODUCT_TYPE_UNDEFINED
   ModalOperationSupported    : 1  = 0x00
   ProductType_UFP_Cable      : 3  = PD_PRODUCT_TYPE_UNDEFINED
   DataCapableAsUSBDevice     : 1  = 0x00
   DataCapableAsUSBHost       : 1  = 0x00
}
```

### 2.40.2 PD_VDM_Discover_Identity_Cert_Stat_VDO

```
Packet PD_VDM_Discover_Identity_Cert_Stat_VDO : PD_DiscoverIdentity_VDO
{
   CertStatVDO_XID       : 32 = 0x00
```

```
}
```

### 2.40.3 PD_VDM_Discover_Identity_Product_VDO

```
Packet PD_VDM_Discover_Identity_Product_VDO : PD_DiscoverIdentity_VDO
{
   ProductVDO_BCDDevice          : 16 = 0x00
   ProductVDO_USBProductId       : 16 = 0x00
}
```

### 2.40.4 PD_VDM_Discover_Identity_Cable_VDO

It's only applicable to Power Delivery Rev 2.

```
Packet PD_VDM_Discover_Identity_Cable_VDO : PD_DiscoverIdentity_VDO
{
   CableVDO_USBSuperSpeedSignalingSupport      : 3 = PD_CABLE_USB31_GEN1_SIGNALING
   CableVDO_SOPDPrimeControllerPresent         : 1 = 0x00
   CableVDO_VBusThroughCable                   : 1 = 0x00
   CableVDO_VBusCurrentHandlingCapability      : 2 = PD_CABLE_CUR_HANDLING_CAP_3A
   CableVDO_SSRX2DirectionalitySupport         : 1 = 0x00
   CableVDO_SSRX1DirectionalitySupport         : 1 = 0x00
   CableVDO_SSTX2DirectionalitySupport         : 1 = 0x00
   CableVDO_SSTX1DirectionalitySupport         : 1 = 0x00
   CableVDO_CableTerminationType               : 2 = 0x00
   CableVDO_CableLatency                       : 4 = PD_CABLE_LATENCY_MAX_10ns
   CableVDO_Rsvd_2                             : 1 = 0x00
   CableVDO_TypeCPlugToTypeA_B_C_Captive       : 2 = PD_CABLE_TYPEC_TO_TYPEC
   CableVDO_Reserved                           : 4 = 0x00
   CableVDO_FirmwareVersion                    : 4 = 0x00
   CableVDO_HardwareVersion                    : 4 = 0x00
}
```

### 2.40.5 PD_VDM_DiscoverIdentity_UFP1_VDO

```
Packet PD_VDM_DiscoverIdentity_UFP1_VDO : PD_DiscoverIdentity_VDO
{
   USBHighestSpeed      : 3  = PD_USB_HIGHEST_SPEED_USB20_ONLY
   AlternateModes       : 3  = PD_SUPP_ALTERNATE_MODE
   Reserved_1           : 18 = 0x00
   DeviceCapability     : 4  = PD_DEV_CAP_USB20
   Reserved_2           : 1  = 0x00
   UFPVDOVersion        : 3  = PD_UFP_VDO_VERSION_1_0
}
```

### 2.40.6 PD_VDM_DiscoverIdentity_UFP2_VDO

```
Packet PD_VDM_DiscoverIdentity_UFP2_VDO : PD_DiscoverIdentity_VDO
{
   USB3MaxPower      : 7 = 0x00
```

```
  USB3MinPower      : 7 = 0x00
  Reserved_1        : 2 = 0x00
  USB4MaxPower      : 7 = 0x00
  USB4MinPower      : 7 = 0x00
  Reserved_2        : 2 = 0x00
}
```

### 2.40.7 PD_VDM_DiscoverIdentity_DFP_VDO

```
Packet PD_VDM_DiscoverIdentity_DFP_VDO : PD_DiscoverIdentity_VDO
{
  PortNumber        : 5  = 0x00
  Reserved_1        : 19 = 0x00
  HostCapability    : 3  = PD_HOST_CAP_USB20
  Reserved_2        : 2 = 0x00
  DFPVDOVersion     : 3  = PD_DFP_VDO_VERSION_1_0
}
```

### 2.40.8 PD_DiscoverIdPassiveCableVdo

```
Packet PD_DiscoverIdPassiveCableVdo : PD_VDM_Discover_Identity_Cable_VDO
{
  USBHighestSpeed       : 3 = PD_USB_HIGHEST_SPEED_USB32_GEN1
  Reserved_1            : 2 = 0x00
  VBusCurHandlingCap    : 2 = PD_CABLE_CUR_HANDLING_CAP_3A
  Reserved_2            : 2 = 0x00
  MaxVBusVoltage        : 2 = PD_CABLE_MAX_VBUS_20V
  CableTerminationType  : 2 = PD_CABLE_VCONN_NOT_REQUIRED
  CableLatency          : 4 = PD_CABLE_LATENCY_MAX_10ns
  EPRModeCapable        : 1 = 0x00
  TypeCtoTypeC_Captive  : 2 = PD_CABLE_TYPEC_TO_TYPEC
  Reserved_3            : 1 = 0x00
  Version               : 3 = PD_CABLE_PASSIVE_VDO_VERSION_1
  FirmwareVersion       : 4 = 0x00
  HardwareVersion       : 4 = 0x00
}
```

### 2.40.9 PD_DiscoverIdActiveCableVdo_1

```
Packet PD_DiscoverIdActiveCableVdo_1 : PD_VDM_Discover_Identity_Cable_VDO
{
  USBHighestSpeed           : 3 = PD_USB_HIGHEST_SPEED_USB32_GEN1
  SOPDoublePrimeController  : 1 = 0x00
  VBusThrough               : 1 = 0x01
  VBusCurHandlingCap        : 2 = PD_CABLE_CUR_HANDLING_CAP_3A
  SBUType                   : 1  = PD_CABLE_PASSIVE_SBU
  SBUSupported              : 1  = PD_FALSE
  MaxVBusVoltage            : 2  = PD_CABLE_MAX_VBUS_20V
  CableTerminationType      : 2  = PD_CABLE_TERM_ACTIVE_ACTIVE
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
   CableLatency                : 4  = PD_CABLE_LATENCY_MAX_10ns
   EPRModeCapable              : 1 = 0x00
   ConnectorType               : 2  = PD_CABLE_TYPEC_TO_TYPEC
   Reserved_1                  : 1 = 0x00
   Version                     : 3  = PD_CABLE_ACTIVE_VDO_VERSION_1_3
   FirmwareVersion             : 4 = 0x00
   HardwareVersion             : 4 = 0x00
}
```

### 2.40.10 PD_DiscoverIdActiveCableVdo_2

```
Packet PD_DiscoverIdActiveCableVdo_2 : PD_VDM_Discover_Identity_Cable_VDO
{
   USBGen                      : 1 = PD_CABLE_ACTIVE_USB_GEN1
   USB4AsymModeSupported       : 1 = PD_FALSE
   OpticallyIsolatedActCbl     : 1 = PD_FALSE
   USBLanesSupported           : 1 = PD_CABLE_ACTIVE_SS_ONE_LANE
   USB32Supported              : 1 = PD_FALSE
   USB2Supported               : 1 = PD_FALSE
   USB2HubHopsConsumed         : 2 = 0x00
   USB4Supported               : 1 = PD_FALSE
   ActiveElement               : 1 = PD_CABLE_ACTIVE_REDRIVER
   PhysicalConnection          : 1 = PD_CABLE_ACTIVE_PHYSICAL_CONN_COPPER
   U3ToU0TransMode             : 1 = PD_CABLE_ACTIVE_U3_TO_U0_DIRECT
   U3CLdPower                  : 3 = PD_CABLE_ACTIVE_U3POWER_GR_THAN_10mW
   Reserved_3                  : 1 = 0x00
   ShutdownTemperature         : 8 = 0x00
   MaxOperatingTemperature     : 8 = 0x00
}
```

### 2.40.11 PD_VDM_Discover_Identity_Alternate_Mode_Adapter_VDO

#### 2.40.11.1 Revision 2.0

```
Packet PD_VDM_Discover_Identity_Alternate_Mode_Adapter_VDO : PD_DiscoverIdentity_VDO
{
   AMDVDO_USBSuperSpeedSignalingSupport    : 3  = 0x01
   AMDVDO_VBusRequired                     : 1  = 0x00
   AMDVDO_VConnRequired                    : 1  = 0x00
   AMDVDO_VConnPower                       : 3  = 0x00
   AMDVDO_SSRX2DirectionalitySupport       : 1  = 0x00
   AMDVDO_SSRX1DirectionalitySupport       : 1  = 0x00
   AMDVDO_SSTX2DirectionalitySupport       : 1  = 0x00
   AMDVDO_SSTX1DirectionalitySupport       : 1  = 0x00
   AMDVDO_Reserved                         : 12 = 0x00
   AMDVDO_FirmwareVersion                  : 4  = 0x00
   AMDVDO_HardwareVersion                  : 4  = 0x00
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
}
```

#### 2.40.11.2 Revision 3.0

```
Packet PD_VDM_Discover_Identity_Alternate_Mode_Adapter_VDO : PD_DiscoverIdentity_VDO
{
   USBHighestSpeed    : 3  = PD_AMA_USB_HIGHEST_SPEED_USB20_ONLY
   VBusRequired       : 1  = 0x00
   VConnRequired      : 1  = 0x00
   VConnPower         : 3  = PD_AMA_VCONN_POWER_1
   Reserved           : 13 = 0x00
   Version            : 3  = PD_AMA_VDO_VERSION_1
   FirmwareVersion    : 4 = 0x00
   HardwareVersion    : 4 = 0x00
}
```

### 2.40.12 PD_DiscoverIdVConnPoweredDeviceVdo

```
Packet PD_DiscoverIdVConnPoweredDeviceVdo : PD_DiscoverIdentity_VDO
{
   ChargeThroughSupport          : 1 = 0x00
   GroundImpedance_1mOUnits      : 6 = 0x00
   VBUSImpedance_2mOUnits        : 6 = 0x00
   VPD_Rsvd_1                    : 1 = 0x00
   ChargeThroughCurrentSupport   : 1 = PD_VPD_CHARGE_THROUGH_CURRENT_SUPP_3A
   MaxVbusVoltage                : 2 = PD_VPD_MAX_VBUS_20V
   VPD_Rsvd_2                    : 4 = 0x00
   VDOVersion                    : 3 = PD_VPD_VDO_VERSION_1
   FirmwareVersion               : 4 = 0x00
   HWVersion                     : 4 = 0x00
}
```

## 2.41 PD_VDM_Discover_Svids_Message

```
Packet PD_VDM_Discover_Svids_Message : PD_VDM_Structured_Header
{
   VDMCommand      = PD_VDM_COMMAND_DISCOVER_SVIDS
   VDMSVID         = PD_VDM_SID
}
```

## 2.42 PD_VDM_Discover_Svids_Response

```
Packet PD_VDM_Discover_Svids_Response : PD_VDM_Discover_Svids_Message
{
   VDMCommandType                = PD_VDM_COMMAND_TYPE_RESPONDER_ACK
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
  DiscoverSVIDsResponderVDOs      : *      Contains one or more VDOs. The only VDO type
                                           which can assign to this field is:
                                           Discover_SVIDs_Responder_VDO

}
```

### 2.42.1 Discover_SVIDs_Responder_VDO

```
Packet Discover_SVIDs_Responder_VDO : PD_Generic_VDO
{
  SVID1   : 16 = 0x00
  SVID2   : 16 = 0x00
}
```

## 2.43 PD_VDM_Discover_Modes_Message

```
Packet PD_VDM_Discover_Modes_Message : PD_VDM_Structured_Header
{
  VDMCommand    = PD_VDM_COMMAND_DISCOVER_MODES
  VDMSVID       = PD_VDM_SID
}
```

## 2.44 PD_VDM_Discover_Modes_Response

```
Packet PD_VDM_Discover_Modes_Response : PD_VDM_Discover_Modes_Message
{
  VDMCommandType    = PD_VDM_COMMAND_TYPE_RESPONDER_ACK

  DiscoverModes     : *      This field should contain one or more VDOs(modes). Each Mode can be a PD_VDO
                             packet variable which has 32bits data length
}
```

### 2.44.1 PD_VDO

```
Packet PD_VDO : PD_Generic_VDO
{
  Data   : 32
}
```

### 2.44.2 PD_VDM_DisplayPort_DiscoverMode_Vdo

```
Packet PD_VDM_DisplayPort_DiscoverMode_Vdo : PD_Generic_VDO
{
  PortCapability                : 2  = PD_DISPLAYPORT_UFPD_CAPABLE
  Signaling                     : 4 = 0x01
  ReceptacleIndication          : 1 = 0x00
  Usb2SignalingNotUsed          : 1 = 0x01
  DFPDPinAssignmentSupported    : 8 = 0x00
```

```
    UFPDPinAssignmentSupported    : 8 = 0x00
    Reserved_DMV                  : 6 = 0x00
    DPAMVersion                   : 2 = 0x00
}
```

## 2.45 PD_VDM_Enter_Mode_Message

```
Packet PD_VDM_Enter_Mode_Message : PD_VDM_Structured_Header
{
    VDMCommand      = PD_VDM_COMMAND_ENTER_MODE
    VDMSVID         = PD_VDM_SID

    VDO             : *     This field may contain one VDO. The VDO can be a
                            PD_VDO packet variable which has 32bits data length.
}
```

## 2.46 PD_VDM_Enter_Mode_Response

```
Packet PD_VDM_Enter_Mode_Response : PD_VDM_Structured_Header
{
    VDMCommand          = PD_VDM_COMMAND_ENTER_MODE
    VDMSVID             = PD_VDM_SID

    VDMCommandType      = PD_VDM_COMMAND_TYPE_RESPONDER_ACK
}
```

## 2.47 PD_VDM_Exit_Mode_Message

```
Packet PD_VDM_Exit_Mode_Message : PD_VDM_Structured_Header
{
    VDMCommand      = PD_VDM_COMMAND_EXIT_MODE
    VDMSVID         = PD_VDM_SID
}
```

## 2.48 PD_VDM_Exit_Mode_Response

```
Packet PD_VDM_Exit_Mode_Response : PD_VDM_Exit_Mode_Message
{
    VDMCommandType    = PD_VDM_COMMAND_TYPE_RESPONDER_ACK
}
```

## 2.49 PD_VDM_Attention_Message

```
Packet PD_VDM_Attention_Message : PD_VDM_Structured_Header
{
    VDMCommand      = PD_VDM_COMMAND_ATTENTION
    VDMSVID         = PD_VDM_SID
```

```
  VDO              : *      This field may contain one VDO. The VDO can be a
                            PD_VDO packet variable which has 32bits data length.
}
```

## 2.50 PD_VDM_DisplayPort_UpdateStatus_Message

```
Packet PD_VDM_DisplayPort_UpdateStatus_Message : PD_VDM_Structured_Header
{
  NumberOfDataObjects   = 0x02

  VDMCommand            = PD_VDM_COMMAND_DISPLAYPORT_STATUS_UPDATE
  VDMSVID               = PD_DISPLAY_PORT_SVID

  StatusVdo             : 32 = 0x00   Contains only one VDO in type of
                                      PD_VDM_DisplayPort_Status_VDO packet template.
}
```

### 2.50.1 PD_VDM_DisplayPort_Status_VDO

```
Packet PD_VDM_DisplayPort_Status_VDO : PD_Generic_VDO
{
  DFPD_UFPD_Connected   : 2  = PD_DISPLAYPORT_DISCONNECTED
  PowerLow              : 1  = 0x00
  AdaptorEnabled        : 1  = 0x00
  MultiFunctionPreferred : 1 = 0x00
  UsbConfigurationRequest : 1 = 0x00
  ExitDisplayModeRequest : 1 = 0x00
  HPD_State             : 1  = 0x00
  IRQ_HPD               : 1  = 0x00
  NO_DPAM_SUSPEND       : 1   = 0x00
  Reserved_DPS_1        : 22 = 0x00
}
```

## 2.51 PD_VDM_DisplayPort_UpdateStatus_Response

```
Packet PD_VDM_DisplayPort_UpdateStatus_Response : PD_VDM_Structured_Header
{
  NumberOfDataObjects   = 0x02

  VDMCommand            = PD_VDM_COMMAND_DISPLAYPORT_STATUS_UPDATE
  VDMCommandType        = PD_VDM_COMMAND_TYPE_RESPONDER_ACK
  VDMSVID               = PD_DISPLAY_PORT_SVID

  StatusVdo             : 32 = 0x00   Contains only one VDO in type of
                                      PD_VDM_DisplayPort_Configure_VDO
                                      packet template.
}
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 2.52 PD_VDM_DisplayPort_Configure_Message

```
Packet PD_VDM_DisplayPort_Configure_Message : PD_VDM_Structured_Header
{
   NumberOfDataObjects     = 0x02

   VDMCommand              = PD_VDM_COMMAND_DISPLAYPORT_CONFIGURE
   VDMSVID                 = PD_DISPLAY_PORT_SVID

   ConfigureVdo            : 32 = 0x00      Contains only one VDO in type of
                                            PD_VDM_DisplayPort_Configure_V
                                            DO packet template
}
```

### 2.52.1 PD_VDM_DisplayPort_Configure_VDO

```
Packet PD_VDM_DisplayPort_Configure_VDO : PD_Generic_VDO
{
   SelectConfiguration     : 2  = PD_DISPLAYPORT_CONFIGURATION_USB
   Signaling               : 4  = 0x00
   Reserved_DPC_1          : 2  = 0x00
   UFPU_PinAssignment      : 8  = 0x00
   Reserved_DPC_2          : 10 = 0x00
   CableUHBR13p5Support    : 1  = 0x00
   Reserved_DPC_3          : 1  = 0x00
   CableType               : 2  = 0x00
   DPAMVersion             : 2  = 0x00
}
```

## 2.53 PD_VDM_DisplayPort_Configure_Response

```
Packet PD_VDM_DisplayPort_Configure_Response : PD_VDM_Structured_Header
{
   VDMCommand              = PD_VDM_COMMAND_DISPLAYPORT_CONFIGURE
   VDMCommandType          = PD_VDM_COMMAND_TYPE_RESPONDER_ACK
   VDMSVID                 = PD_DISPLAY_PORT_SVID
}
```

## 2.54 PD_ExtendedMsgHeader

```
Packet PD_ExtendedMsgHeader : PD_Packet
{
   DataSize          : 9 = 0x00
   Reserved          : 1 = 0x00
   RequestChunk      : 1 = 0x00
   ChunkNumber       : 4 = 0x00
   Chunked           : 1 = 0x00
}
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 2.55 PD_ExtMsgHeaders

```
Packet PD_ExtMsgHeaders : PD_Packet
{
   : PD_MessageHeader
   : PD_ExtendedMsgHeader

   Extended                    = 1
}
```

## 2.56 PD_SourceCapExtendedMsg

```
Packet PD_SourceCapExtendedMsg : PD_ExtMsgHeaders
{
   MessageType                 = 1
   DataSize                    = PD_SCEDB_DATA_SIZE

   : PD_SourceCapExtDataBlock
}
```

### 2.56.1 PD_SourceCapExtDataBlock

```
Packet PD_SourceCapExtDataBlock
{
   VendorId                    : 16 = 0x00
   ProductId                   : 16 = 0x00
   XId                         : 32 = 0x00
   FirmwareVersion             : 8  = 0x00
   HardwareVersion             : 8  = 0x00
   LoadStep                    : 2  = 0x00
   IOC                         : 1  = 0x00
   Reserved_1                  : 5  = 0x00
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
  HoldupTime                          : 8  = 0x00
  LPSCompliant                        : 1  = 0x00
  PS1Compliant                        : 1  = 0x00
  PS2Compliant                        : 1  = 0x00
  Reserved_2                          : 5  = 0x00
  LowTouchCurEPS                      : 1  = 0x00
  GroundPinSupport                    : 1  = 0x00
  GrndPinForProtectiveEarth           : 1  = 0x00
  Reserved_3                          : 5  = 0x00
  PeakCur1_PercentOverload            : 5  = 0x00
  PeakCur1_OverloadPeriod             : 6  = 0x00
  PeakCur1_DutyCycle                  : 4  = 0x00
  PeakCur1_VbusVoltageDroop           : 1  = 0x00
  PeakCur2_PercentOverload            : 5  = 0x00
  PeakCur2_OverloadPeriod             : 6  = 0x00
  PeakCur2_DutyCycle                  : 4  = 0x00
  PeakCur2_VbusVoltageDroop           : 1  = 0x00
  PeakCur3_PercentOverload            : 5  = 0x00
  PeakCur3_OverloadPeriod             : 6  = 0x00
  PeakCur3_DutyCycle                  : 4  = 0x00
  PeakCur3_VbusVoltageDroop           : 1  = 0x00
  TouchTemp                           : 8  = 0x00
  ExternalSupplyIsPresent             : 1  = 0x00
  ExternalSupplyCondition             : 1  = 0x00
  InternalBatteryIsPresent            : 1  = 0x00
  Reserved_4                          : 5  = 0x00
  NumberOfFixedBatteries              : 4  = 0x00
  NumberOfHotSwappableBatterySlots    : 4  = 0x00
  SourcePDP                           : 7  = 0x00
  Reserved_5                          : 1  = 0x00
  EPRSourcePDP                        : 8  = 0x00
}
```

## 2.57 PD_StatusMsg

```
Packet PD_StatusMsg : PD_ExtMsgHeaders
{
  MessageType           = 2
  DataSize              = PD_SOP_SDB_DATA_SIZE

  : PD_StatusDataBlock
}
```

### 2.57.1 PD_StatusDataBlock

```
Packet PD_StatusDataBlock
{
   InternalTemp                        : 8 = 0x00

   PresentInput_Rsvd_1                 : 1 = 0x00
   ExternalPowerIsPresent              : 1 = 0x00
   ExternalPower_AC_DC                 : 1 = 0x00
   InternalPowerBattery                : 1 = 0x00
   InternalPowerNonBattery             : 1 = 0x00
   PresentInput_Rsvd_1_2               : 3 = 0x00

   FixedBattery                        : 4 = 0x00
   HotSwappableBattery                 : 4 = 0x00

   EventFlags_Rsvd_1                   : 1 = 0x00
   OverCurProtectionEvent              : 1 = 0x00
   OverTempProtectionEvent             : 1 = 0x00
   OverVoltProtection                  : 1 = 0x00
   OperatingModeFlag                   : 1 = 0x00
   EventFlags_Rsvd_2                   : 3 = 0x00

   TempStatus_Rsvd_1                   : 1 = 0x00
   TemperatureStatus                   : 2 = PD_TEMP_STATUS_NORMAL
   TempStatus_Rsvd_2                   : 5 = 0x00

   PowerStatus_Rsvd_1                  : 1 = 0x00
   PowLimited_CableCurrent             : 1 = PD_FALSE
   PowLimited_InsuffPower              : 1 = PD_FALSE
   PowLimited_InsuffExtPower           : 1 = PD_FALSE
   PowLimited_EventFlagsSet            : 1 = PD_FALSE
   PowLimited_Temperature              : 1 = PD_FALSE
   PowerStatus_Rsvd_2                  : 2 = 0x00
   PowerStateChange_NewState           : 3 = 0x00
   PowerStateChange_NewStateIndicator  : 3 = 0x00
   PowerStateChange_Rsvd_1             : 2 = 0x00
}
```

## 2.58 PD_StatusMsg_Cable

```
Packet PD_StatusMsg_Cable : PD_ExtMsgHeaders
{
   MessageType                 = 2
   DataSize                    = PD_SOPP_SDB_DATA_SIZE

   : PD_StatusDataBlock_Cable
}
```

### 2.58.1 PD_StatusDataBlock_Cable

```
Packet PD_StatusDataBlock_Cable
{
   InternalTemp          : 8 = 0x00
   ThermalShutdown       : 1 = PD_FALSE
   Rsvd                  : 7 = 0x00
}
```

## 2.59 PD_GetBatteryCapMsg

```
Packet PD_GetBatteryCapMsg : PD_ExtMsgHeaders
{
   MessageType                    = 3
   DataSize                       = PD_GBCDB_DATA_SIZE

   : PD_GetBatteryCapDataBlock
}
```

### 2.59.1 PD_GetBatteryCapDataBlock

```
Packet PD_GetBatteryCapDataBlock
{
   BatteryCapRef     : 8 = 0x00
}
```

## 2.60 PD_GetBatteryStatusMsg

```
Packet PD_GetBatteryStatusMsg : PD_ExtMsgHeaders
{
   MessageType                    = 4
   DataSize                       = PD_GBSDB_DATA_SIZE

   : PD_GetBatteryStatusDataBlock
}
```

### 2.60.1 PD_GetBatteryStatusDataBlock

```
Packet PD_GetBatteryStatusDataBlock
{
   BatteryStatusRef    : 8 = 0x00
}
```

## 2.61 PD_BatteryCapabilitiesMsg

```
Packet PD_BatteryCapabilitiesMsg : PD_ExtMsgHeaders
{
   MessageType                = 5
   DataSize                   = PD_BCDB_DATA_SIZE

   : PD_BatteryCapDataBlock
}
```

### 2.61.1 PD_BatteryCapDataBlock

```
Packet PD_BatteryCapDataBlock
{
   VendorId                          : 16 = 0x00
   ProductId                         : 16 = 0x00
   DesignCap_100mWHUnits             : 16 = 0x00
   LastFullChargeCap_100mWHUnits     : 16 = 0x00
   InvalidBatteryRef                 : 1  = 0x00
   BCDB_Rsvd_1                       : 7  = 0x00
}
```

## 2.62 PD_GetManufacturerInfoMsg

```
Packet PD_GetManufacturerInfoMsg : PD_ExtMsgHeaders
{
   MessageType                = 6
   DataSize                   = PD_GMIDB_DATA_SIZE

   : PD_GetManufacturerInfoDataBlock
}
```

### 2.62.1 PD_GetManufacturerInfoDataBlock

```
Packet PD_GetManufacturerInfoDataBlock
{
   Target                : 8 = PD_MANINFO_TARGET_PORT_CABLE
   ManufacturerInfoRef   : 8 = 0x00
}
```

## 2.63 PD_ManufacturerInfoMsg

```
Packet PD_ManufacturerInfoMsg : PD_ExtMsgHeaders
{
   MessageType                = 7
   DataSize                   = PD_MIDB_DATA_SIZE

   : PD_ManufacturerInfoDataBlock
```

```
}
```

### 2.63.1 PD_ManufacturerInfoDataBlock

```
Packet PD_ManufacturerInfoDataBlock
{
   VendorId              : 16 = 0x00
   ProductId             : 16 = 0x00
   ManufacturerString    : *          Can be initialized using a byte stream (max is 176b)
}
```

## 2.64 PD_SecurityRequestMsg

```
Packet PD_SecurityRequestMsg : PD_ExtMsgHeaders
{
   MessageType           = 8
   DataSize              = PD_SRQDB_DATA_SIZE

   SecurityRequestDB     : *    Can contain only one Security Request Data Block. Available SRDB
                                templates are:
                                PD_SRQDB_GetDigests,
                                PD_SRQDB_GetCertificate,
                                PD_SRQDB_Challenge
}
```

### 2.64.1 PD_SecurityDBHeader

```
Packet PD_SecurityDBHeader
{
   AuthProtocolVersion   : 8 = PD_AUTH_PROT_VER_1
   AuthMessageType       : 8 = 0x00
   AuthParam1            : 8 = 0x00
   AuthParam2            : 8 = 0x00
}
```

### 2.64.2 PD_SecurityRequestDB

```
Packet PD_SecurityRequestDB : PD_SecurityDBHeader
{
}
```

### 2.64.3 PD_SRQDB_GetDigests

```
Packet PD_SRQDB_GetDigests : PD_SecurityRequestDB
{
   AuthMessageType    = PD_AUTH_TYPE_GET_DIGESTS
}
```

### 2.64.4 PD_SRQDB_GetCertificate

```
Packet PD_SRQDB_GetCertificate : PD_SecurityRequestDB
{
   AuthMessageType    = PD_AUTH_TYPE_GET_CERTIFICATE

   Offset             : 16 = 0x00
   Length             : 16 = 0x00
}
```

### 2.64.5 PD_SRQDB_Challenge

```
Packet PD_SRQDB_Challenge : PD_SecurityRequestDB
{
   AuthMessageType    = PD_AUTH_TYPE_GET_CHALLENGE

   Nonce              : 256 = { 00 00 00 00 }
}
```

## 2.65 PD_SecurityResponseMsg

```
Packet PD_SecurityResponseMsg : PD_ExtMsgHeaders
{
   MessageType        = 9
   DataSize           = PD_SRPDB_DATA_SIZE

   SecurityResponseDB : *
}
```

Can contain only one Security Response Data Block. Available SRPDB types are:
`PD_SRPDB_Digests,`
`PD_SRPDB_Certificate,`
`PD_SRPDB_ChallengeAuth,`
`PD_SRPDB_Error`

### 2.65.1 PD_SecurityResponseDB

```
Packet PD_SecurityResponseDB : PD_SecurityDBHeader
{
}
```

### 2.65.2 PD_SRPDB_Digests

```
Packet PD_SRPDB_Digests : PD_SecurityResponseDB
{
   AuthMessageType    = PD_AUTH_TYPE_DIGESTS
   AuthParam1         = 0x01
```

```
  DigestArray          : *
}
```

Max len is 256 bytes, each digest is 32 bytes. Packet variables of `PD_Security_Digest` type can be assigned to this field.

#### 2.65.2.1 PD_Security_Digest

```
Packet PD_Security_Digest
{
  Digest      : 256 = 0x00
}
```

### 2.65.3 PD_SRPDB_Certificate

```
Packet PD_SRPDB_Certificate : PD_SecurityResponseDB
{
  AuthMessageType    = PD_AUTH_TYPE_CERTIFICATE

  Certificate        : *
}
```

Can be initialized using a byte stream.

### 2.65.4 PD_SRPDB_ChallengeAuth

```
Packet PD_SRPDB_ChallengeAuth : PD_SecurityResponseDB
{
  AuthMessageType     = PD_AUTH_TYPE_CHALLENGE_AUTH

  MinProtVer          : 8 = 0x00
  MaxProtVer          : 8 = 0x00
  Capabilities        : 8 = 0x01
  Rsvd                : 8 = 0x00
  CertChainHash       : 256 = { 00 00 00 00 }
  Salt                : 256 = { 00 00 00 00 }
  ContextHash         : 256 = { 00 00 00 00 }
  Signature           : 512 = { 00 00 00 00 }
}
```

### 2.65.5 PD_SRPDB_Error

```
Packet PD_SRPDB_Error : PD_SecurityResponseDB
{
  AuthMessageType     = PD_AUTH_TYPE_ERROR
}
```

## 2.66 PD_PPSStatusMsg

```
Packet PD_PPSStatusMsg : PD_ExtMsgHeaders
{
   MessageType                = 12
   DataSize                   = PD_PPSSDB_DATA_SIZE

   : PD_PPSStatusDataBlock
}
```

### 2.66.1 PD_PPSStatusDataBlock

```
Packet PD_PPSStatusDataBlock
{
   OutputVoltage_20mVUnits    : 16 = 0xFFFF
   OutputCurrent_50mAUnits    : 8  = 0xFF
   PPSSDB_Rsvd_1              : 1  = 0x00
   PresentTemperatureFlag     : 2  = 0x01
   OperatingModeFlag          : 1  = 0x00
   PPSSDB_Rsvd_2              : 4  = 0x00
}
```

## 2.67 PD_CountryInfoMsg

```
Packet PD_CountryInfoMsg : PD_ExtMsgHeaders
{
   MessageType                = 13
   DataSize                   = PD_CIDB_DATA_SIZE

   : PD_CountryInfoDataBlock
}
```

### 2.67.1 PD_CountryInfoDataBlock

```
Packet PD_CountryInfoDataBlock
{
   : CountryCode

   CIDB_Rsvd_1          : 16 = 0x00

   CountrySpecificData  : *          Byte stream data. Max length is 256Bytes
}
```

## 2.68 PD_CountryCodesMsg

```
Packet PD_CountryCodesMsg : PD_ExtMsgHeaders
{
   MessageType                = 14
   DataSize                   = PD_CCDB_DATA_SIZE
```

```
    : PD_CountryCodesDataBlock
}
```

### 2.68.1 PD_CountryCodesDataBlock

```
Packet PD_CountryCodesDataBlock
{
   NumberOfCountryCodes    : 16 = 0x01

   : CountryCode

   CountryCodes            : *     Should be multiple of 16bits.
                                   Max Len is 256Bytes.
                                   Each CountryCode is 2Bytes.
                                   Each CountryCode can be assigned using the
                                   CountryCode packet variable.
}
```

#### 2.68.1.1 CountryCode

```
Packet CountryCode
{
   Alpha2CountryCode1stChar    : 8 = 0x00
   Alpha2CountryCode2ndChar    : 8 = 0x00
}
```

## 2.69 PD_SinkCapExtendedMsg

```
Packet PD_SinkCapExtendedMsg : PD_ExtMsgHeaders
{
   MessageType             = 15
   DataSize                = PD_SKEDB_DATA_SIZE

   : PD_SinkCapExtDataBlock
}
```

### 2.69.1 PD_SinkCapExtDataBlock

```
Packet PD_SinkCapExtDataBlock
{
   VendorId                : 16 = 0x00
   ProductId               : 16 = 0x00
   XId                     : 32 = 0x00
   FirmwareVersion         : 8  = 0x00
```

```
    HardwareVersion                 : 8  = 0x00
    SKEDBVersion                    : 8  = PD_SKEDB_VERSION_1
    LoadStep                        : 2  = PD_SKEDB_LOADSTEP_150mA_PER_uS
    LoadStep_Rsvd                   : 6  = 0x00
    OverLoad_10PercentUnits         : 5  = 0x00
    OverloadPeriod_20mSUnits        : 6  = 0x00
    DutyCycle_5PercentUnits         : 4  = 0x00
    TolerateVBusDroop               : 1  = 0x00
    RequiresLPSSource               : 1  = 0x00
    RequiresPS1Source               : 1  = 0x00
    RequiresPS2Source               : 1  = 0x00
    Compliance_Rsvd                 : 5  = 0x00
    TouchTemp                       : 8  = PD_SKEDB_TOUCHTEMP_DEFAULT
    NumberOfFixedBatteries          : 4  = 0x00
    NumberOfHotSwappableSlots       : 4  = 0x00
    PPSChargingSupported            : 1  = 0x00
    VBusPowered                     : 1  = 0x01
    MainsPowered                    : 1  = 0x00
    BatteryPowered                  : 1  = 0x00
    BatteryEssentiallyUnlimited     : 1  = 0x00
    AVSSupport                      : 1  = 0x00
    SinkModes_Rsvd                  : 2  = 0x00
    SinkMinPDP                      : 7  = 0x00
    SinkMinPDP_Rsvd                 : 1  = 0x00
    SinkOperationalPDP              : 7  = 0x00
    SinkOperationalPDP_Rsvd         : 1  = 0x00
    SinkMaxPDP                      : 7  = 0x00
    SinkMaxPDP_Rsvd                 : 1  = 0x00
    EPRSinkMinPDP                   : 8  = 0x00
    EPRSinkOperationalPDP           : 8  = 0x00
    EPRSinkMaxPDP                   : 8  = 0x00
}
```

## 2.70 PD_GetSourceInfoMsg

```
Packet PD_GetSourceInfoMsg: PD_ControlMessage
{
    MessageType     = PD_MESSAGE_TYPE_GET_SOURCE_INFO
}
```

## 2.71 PD_GetRevisionMsg

```
Packet PD_GetRevisionMsg: PD_ControlMessage
{
    MessageType     = PD_MESSAGE_TYPE_GET_REVISION
}
```

## 2.72 PD_EPRModePacket

```
Packet PD_EPRModePacket : PD_MessageHeader
{
   MessageType              = 10
   NumberOfDataObjects      = 1

   : PD_EPRModeData
}
```

### 2.72.1 PD_EPRModeData

```
Packet PD_EPRModeData
{
   Rsvd     : 16 = 0
   Data     : 8  = 0
   Action   : 8  = 0
}
```

## 2.73 PD_ExtendedCtrlMsg

```
Packet PD_ExtendedCtrlMsg : PD_ExtMsgHeaders
{
   MessageType                          = 16
   DataSize                             = 2

   : PD_ExtControlDataBlock
}
```

### 2.73.1 PD_ExtControlDataBlock

```
Packet PD_ExtControlDataBlock
{
   type     : 8 = 0x00
   data     : 8 = 0x00
}
```

## 2.74 PD_GetEPRSourceCapMsg

```
Packet PD_GetEPRSourceCapMsg : PD_ExtendedCtrlMsg
{
   type     = 1
   data     = 0
}
```

## 2.75 PD_GetEPRSinkCapMsg

```
Packet PD_GetEPRSinkCapMsg : PD_ExtendedCtrlMsg
{
   type      = 2
   data      = 0
}
```

## 2.76 PD_EPRKeepAliveMsg

```
Packet PD_GetEPRKeepAliveMsg : PD_ExtendedCtrlMsg
{
   type      = 3
   data      = 0
}
```

## 2.77 PD_EPRKeepAliveAckMsg

```
Packet PD_GetEPRKeepAliveAckMsg : PD_ExtendedCtrlMsg
{
   type      = 4
   data      = 0
}
```

## 2.78 PD_EPRSourceCapabilitiesMessage

```
Packet PD_EPRSourceCapabilitiesMessage : PD_ExtendedCtrlMsg
{
   MessageType                = 17
   DataSize                   = 0

   EPRSourceCapabilitiesData  : *
}
```

This field holds SPR and EPR PDOs.
SPR PDO types which can assign to this field are:
`PD_PowerDataObjectFixedSupply_Source`,
`PD_PDOFixedSupplyNotVSafe5V_Source`,
`PD_PowerDataObjectVariableSupply_Source`,
`PD_PowerDataObjectBatterySupply_Source`,
`PD_PowerDataObjectPPS_Source`,
`PD_SPRPowerDataObjectAVS`EPR PDO types which can assign to this field are:
`PD_PowerDataObjectFixedSupply_Source`,
`PD_EPRPowerDataObjectAVS`

### 2.78.1 PD_EPRPowerDataObjectAVS

```
Packet PD_EPRPowerDataObjectAVS : PD_PowerDataObject
{
   PDP_1WUnits                : 8 = 0
   MinVoltage_100mVUnits      : 8 = 0
   AVS_Rsvd_1                 : 1 = 0
   MaxVoltage_100mVUnits      : 9 = 0
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
    PeakCurrent                 : 2 = 0
    APDOType                    : 2 = 1
    PowerDataType               : 2 = 3
}
```

## 2.79 PD_EPRSinkCapabilitiesMessage

```
Packet PD_EPRSinkCapabilitiesMessage : PD_ExtendedCtrlMsg
{
    MessageType             = 18
    DataSize                = 0

    EPRSinkCapabilitiesData : *
}
```

This field holds SPR and EPR PDOs.
SPR PDO types which can assign to this field are:
`PD_PowerDataObjectFixedSupply_Sink,`
`PD_PowerDataObjectVariableSupply_Sink,`
`PD_PowerDataObjectBatterySupply_Sink,`
`PD_PowerDataObjectPPS_Sink,`
`PD_SPRPowerDataObjectAVS`
EPR PDO types which can assign to this field are:
`PD_PowerDataObjectFixedSupply_Sink,`
`PD_EPRPowerDataObjectAVS`

## 2.80 PD_EPRRequestPacket

```
Packet PD_EPRRequestPacket : PD_MessageHeader
{
    MessageType             = 9
    NumberOfDataObjects     = 2
    RDO                     : 32 = 0x00
    PDO                     : 32 = 0x00
}
```

This field can contain only one RDO packet variables.
Available RDO templates are:
`PD_RequestDataObject_Fixed_Variable_NoGiveBack,`
`PD_RequestDataObject_Fixed_Variable_GiveBack,`
`PD_RequestDataObject_Battery_NoGiveBack,`
`PD_RequestDataObject_Battery_GiveBack,`
`PD_ProgrammableRDO`

This field is the selected PDO in the SourceCapabilities packet

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

# 3 Type-C Commands

In addition to Power Delivery commands, PD Exerciser also provides a command set to manage USB Type-C connection. It includes some low level commands for manipulating voltages, capacitors and resistors as well as some high level commands that let you have SINK, SINKAS, SOURCE and DRP state machines, described in Type-C specification, with the facilities to customize different behaviors and characteristics. Note that at the present, Type-C state machines are just followed when related commands are running. In other words, Type-C state machines are not followed in parallel to other Power Delivery commands execution.

## 3.1 PD_SetResistorRp

Sets resistor Rp On/Off.

**Format**

```
Call PD_SetResistorRp( state, current, line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`current`

Possible values:

```
CC_RP_CUR_DEFAULT
CC_RP_CUR_1_5
CC_RP_CUR_3_0
```

`line`

Possible values:

```
CC_LINE_1
CC_LINE_2
CC_LINE_ALL
```

**Result**

None

**Examples**

```
Call PD_SetResistorRP( PD_ON, CC_RP_CUR_1_5, CC_LINE_2 )
```

## 3.2 PD_SetResistorRd

Sets resistor Rd On/Off.

**Format**

```
Call PD_SetResistorRd( state, line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`line`

Possible values:

```
CC_LINE_1
CC_LINE_2
CC_LINE_ALL
```

**Result**

None

**Examples**

```
Call PD_SetResistorRd( PD_ON, CC_LINE_1 )
```

## 3.3 PD_SetResistorRa

Sets resistor Ra On/Off.

**Format**

```
Call PD_SetResistorRa( state, line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`line`

Possible values:

```
CC_LINE_1
CC_LINE_2
CC_LINE_ALL
```

**Result**

None

**Examples**

```
Call PD_SetResistorRa( PD_ON, CC_LINE_2 )
```

## 3.4 PD_SetVBusCap10MicroFarad

Sets the VBus Capacitor(10 Micro Farad) On/Off.

**Format**

```
Call PD_SetVBusCap10MicroFarad( state )
```

**Parameters**

`state`

Possible values:

PD_ON
PD_OFF

### Result

None

### Examples

```
Call PD_SetVBusCap10MicroFarad( PD_ON )
```

## 3.5 PD_SetVBusCap1MicroFarad

Sets the VBus Capacitor(1 Micro Farad) On/Off.

### Format

```
Call PD_SetVBusCap1MicroFarad( state )
```

### Parameters

`state`

Possible values:

PD_ON
PD_OFF

### Result

None

### Examples

```
Call PD_SetVBusCap1MicroFarad( PD_ON )
```

## 3.6 PD_SetVBusSetting

It is used to customize behavior of `PD_SetVBus` command.

### Format

```
Call PD_SetVBusSetting( PD_Vbus_Settings $settings )
```

### Parameters

`$settings`

Parameter type is `PD_Set_Vbus_Settings`. Available fields for this type are:

| Field Names | Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT | Timeout (in microseconds) on waiting for Vbus line voltage to reach to the specified level. |
| LowerVoltageTolerance | PD_SET_VBUS_NO_VOLTAGE_TOLERANCE | The accepted voltage deviation (in millivolt) for returning from `PD_SetVbus` command. The command returns when the voltage gets greater than Desired Voltage – `LowerVoltageTolerance` |
| UpperVoltageTolerance | PD_SET_VBUS_NO_VOLTAGE_TOLERANCE | The accepted voltage deviation (in millivolt) for returning from `PD_SetVbus` command. The command returns when the voltage gets less than Desired Voltage + Upper`VoltageTolerance` |
| AttemptExactVoltage | PD_FALSE | If is set to `PD_TRUE`, the `PD_SetVBus` command repeats setting voltage to get closer to the requested voltage by using feedback from Vbus line. It is not recommended for normal use cases. |

| | | |
|---|---|---|
| AttemptExactVoltageInterval_us | 50000 | The delay (in microseconds) before reading voltage from Vbus line in `AttempExactVoltage process.` |

**Result**

None

**Examples**

```
$vbus_setting = PD_Set_Vbus_Settings
{
   WaitTimeout = 5000000 # 5 seconds

LowerVoltageTolerance = 250 # milliVolt
UpperVoltageTolerance = 500 # milliVolt

}
Call PD_SetVBusSetting( $vbus_setting )

# The next command sets the voltage to 8 Volt and returns as soon as 7.750 <= Voltage <= 8.5 is seen on
the Vbus line
Call PD_SetVBus( PD_ON, 8000 )
```

## 3.7 PD_SetVBus

Sets VBus On/Off. The command is blocked until the voltage get applied to specified level.

**Format**

```
Call PD_SetVBus( state, voltage_milli_volt )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`voltage_milli_volt`

The voltage which applied on VBus. `voltage_mili_volt` cannot be greater than 20500 mV. In order to apply voltages greater than 5V, the corresponding check box should be set in recording options.

**Result**

None

**Examples**

```
Call PD_SetVBus( PD_ON, 5000 )
```

## 3.8 PD_WaitForVBus

Compares VBus voltage with the desired value within a specific time.

**Format**

```
Call PD_WaitForVBus( voltage_mv, wait_timeout_us, compare_operator )
```

**Parameters**

`voltage_mv`

The desired VBus voltage value(in milli-volt) to be compared with the VBus voltage value.

`wait_timeout_us`

The function will monitor the VBus voltage within this timeout (in micro seconds). If VBus voltage reaches to the desired value during this timeout then the function returns `PD_RESULT_OK` otherwise it returns `PD_RESULT_FAILED` with sub-result set to `PD_SUBRESULT_TIMEOUT_OCCURRED`.

`compare_operator`

Indicates the desired comparator.

Possible values:

```
PD_COMPARE_EQUAL
PD_COMPARE_GREATER
PD_COMPARE_LESS
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Values |
|---|
| PD_RESULT_OK |
| PD_RESULT_FAILED |
| PD_SUBRESULT_TIMEOUT_OCCURRED |

**Examples**

```
call PD_WaitForVBus(VBUS_VOLTAGE_V5_SAFE_MIN, 1000000, PD_COMPARE_GREATER)
```

## 3.9 PD_SetVConn

Sets VConn On/Off.

**Format**

```
Call PD_SetVConn( state, cc_line, voltage_mili_volt )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_AUTO
CC_LINE_1
CC_LINE_2
```

`voltage_mili_volt`

Indicates the desired VConn voltage in mV.

**Result**

None

**Examples**

```
Call PD_SetVConn( PD_ON, CC_LINE_2, 3000 )
```

## 3.10 PD_SetLoadOnVBus

Enables/Disables load on VBus.

**Format**

```
Call PD_SetLoadOnVBus( state )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

**Result**

None

**Examples**

```
Call PD_SetLoadOnVBus( PD_ON )
```

## 3.11 PD_SetResistor100K

Sets the Resistor100K on the specified CC line.

**Format**

```
Call PD_SetResistor100K( state, cc_line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_ALL
CC_LINE_1
CC_LINE_2
```

**Result**

None

**Examples**

```
Call PD_SetResistor100K( PD_ON, CC_LINE_1 )
```

## 3.12 PD_SetResistor95_3K

Sets the Resistor95.3K on the specified CC line.

**Format**

```
Call PD_SetResistor95_3K( state, cc_line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_ALL
CC_LINE_1
CC_LINE_2
```

**Result**

None

**Examples**

```
Call PD_SetResistor95_3K( PD_ON, CC_LINE_1 )
```

## 3.13 PD_SetResistor264K

Sets the Resistor264K on the specified CC line.

**Format**

```
Call PD_SetResistor264K( state, cc_line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_ALL
CC_LINE_1
CC_LINE_2
```

**Result**

None

**Examples**

```
Call PD_SetResistor264K( PD_ON, CC_LINE_1 )
```

## 3.14 PD_SetBCSourceMode

Enables/Disables Battery Charger Source Mode.

**Format**

```
Call PD_SetBCSourceMode( state )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

**Result**

None

**Examples**

```
Call PD_SetBCSourceMode( PD_ON )
```

## 3.15 PD_SetCapacitor400pF

Sets the Capacitor400pF on the specified CC line.

**Format**

```
Call PD_SetCapacitor400pF( state, cc_line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_ALL
CC_LINE_1
CC_LINE_2
```

**Result**

None

**Examples**

```
Call PD_SetCapacitor400pF( PD_ON, CC_LINE_1 )
```

## 3.16 PD_SetCC1Capacitor390pF

Enables/Disables the Capacitor 390pF on CC1.

**Format**

```
Call PD_SetCC1Capacitor390pF( state )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

**Result**

None

**Examples**

```
Call PD_SetCC1Capacitor390pF( PD_ON )
```

## 3.17 PD_ReportSafeStateStatus

Enables/Disables USB Safe States report on the specified lines.

**Format**

```
Call PD_ReportSafeStateStatus( lines )
```

**Parameters**

`lines`

Possible values:

```
DP_LINE_1
DP_LINE_2
DM_LINE_1
DM_LINE_2
RX_LINE_1
RX_LINE_2
TX_LINE_1
TX_LINE_2
SBU_LINE_1
SBU_LINE_2
```

**Result**

None

**Examples**

```
#Enables USB Safe States report on DP_LINE_2, DM_LINE_2, SBU_LINE_1, SBU_LINE_2
#lines and disables the report on all other lines.
call PD_ReportSafeStateStatus(DP_LINE_2 | DM_LINE_2 | SBU_LINE_1 | SBU_LINE_2)
```

## 3.18 PD_SetVbusToCCWithResistor53_2K

Puts VBus voltage on specified CC line using the 53.2K resistor

**Format**

```
Call PD_SetVbusToCCWithResistor53_2K( state, cc_line )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

`cc_line`

Possible values:

```
CC_LINE_ALL
CC_LINE_1
CC_LINE_2
```

**Result**

None

**Examples**

```
Call PD_SetVbusToCCWithResistor53_2K( PD_ON, CC_LINE_1 )
```

## 3.19 PD_AllowVbusWithoutCCTerm

It allows/prevents sourcing Vbus without any CC terminations.

**Format**

```
Call PD_AllowVbusWithoutCCTerm( state )
```

**Parameters**

`state`

Possible values:

```
PD_ON
PD_OFF
```

**Result**

None

**Examples**

```
Call PD_AllowVbusWithoutCCTerm( PD_ON )
```

## 3.20 PD_TerminateCCLines

Terminates CC lines with the specified resistors.

**Format**

```
Call PD_TerminateCCLines( CC1_Resistor, CC2_Resistor )
```

**Parameters**

`CC1_Resistor`

Possible values:

```
CC_OPEN
CC_RP
CC_RP_1_5
CC_RP_3_0
CC_RD
CC_RA
```

`CC2_Resistor`

Possible values:

```
CC_OPEN
CC_RP
CC_RP_1_5
CC_RP_3_0
CC_RD
CC_RA
```

**Result**

None

**Examples**

```
Call PD_TerminateCCLines( CC_RP_1_5, CC_OPEN )
```

## 3.21 PD_WaitForUUTPinState

Waits until either a timeout being occurred or a specific UUT pin state being detected for a specific duration of time.

**Format**

```
Call PD_WaitForUUTPinState( pin_state, state_duration, pin_selector, timeout )
```

**Parameters**

`pin_state`

The specific UUT pin state to check.

Possible values:

```
CC_PIN_STATE_NONE
CC_PIN_STATE_SRC_OPEN
CC_PIN_STATE_SRC_RD
CC_PIN_STATE_SRC_RA
CC_PIN_STATE_SNK_OPEN
CC_PIN_STATE_SNK_RP
```

`state_duration`

Time duration(in micro seconds) to check the presentation of the UUT specified pin state.

`pin_selector`

Indicates the CC line(s)

```
CC_PIN_SELECTOR_NEITHER
CC_PIN_SELECTOR_EITHER
CC_PIN_SELECTOR_BOTH
CC_PIN_SELECTOR_ONLY_ONE
CC_PIN_SELECTOR_ONE_OTHER
CC_PIN_SELECTOR_CC1
CC_PIN_SELECTOR_CC2
```

`timeout`

If the timeout(in micro seconds) occurs function will return.

**Result**

None

**Examples**

```
Call PD_WaitForUUTPinState(CC_PIN_STATE_SRC_RD, 50000, CC_PIN_SELECTOR_CC2, 200000)
```

## 3.22 PD_SetStartDRPSetting

It is used to customize behavior and characteristics of the Exerciser when acts as a DRP device. Settings will be applied to PD_StartDRP function.

**Format**

```
Call PD_SetStartDRPSetting( PD_Start_DRP_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is PD_Start_DRP_Settings. This type contains following data fields:

| Field Name | Default Values | Description |
|---|---|---|
| Timeout | PD_DEFAULT_TIMEOUT_INFINIT | Indicates the timeout in micro second for connecting as a SINK or SOURCE. Should be greater than 5000us. |
| WithRa | PD_FALSE | Indicates whether to provide Ra on one of CC lines or not. |
| WithAccessory | PD_FALSE | If set to PD_TRUE, the command checks whether the UUT transitions to the AudioAccessory state or not. |
| AdvertizedCurrent | CC_RP_CUR_1_5 | Indicates advertised current level on Rp. |
| WithVConn | PD_FALSE | Indicates whether to turn on the VConn or not. |
| VConnVoltage_mV | VCONN_VOLTAGE_DEFAULT | Indicates the VConn voltage level when start sourcing. |
| AccessoryStateDuration | 1000000 (us) | Time duration to check whether the UUT is in AudioAccessory state or not. |
| StartWithSNK | PD_FALSE | If is set to PD_TRUE, DRP state machine starts from Unattached.SNK state instead of Unattached.SRC state. |
| WithTrySRC | PD_FALSE | If is set to PD_TRUE, Exerciser supports Try.SRC state machine. |
| WithTrySNK | PD_FALSE | If is set to PD_TRUE, Exerciser supports Try.SNK state machine. |

**Result**

None

**Examples**

```
$startdrp_setting = PD_Start_DRP_Settings
{
   WithTrySRC = PD_TRUE
}
Call PD_SetStartDRPSetting( $startdrp_setting )
```

## 3.23 PD_StartDRP

It starts DRP state machine for connecting to a Type-C device. The command quits if timeouts or Exerciser transitions to Attached.Src or Attached.SNK. Usually is being called after PD_SetStartDRPSetting function.

**Format**

```
Call PD_StartDRP()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Values |
|---|
| PD_RESULT_OK |
| PD_RESULT_FAILED |

**Examples**

```
Call PD_StartDRP()
```

## 3.24 PD_SetStartSourceSetting

It is used to customize behavior and characteristics of the Exerciser when acts as a SOURCE device. Settings will be applied to PD_StartSource command.

### Format

```
Call PD_SetStartSourceSetting( PD_Start_Source_Settings $settings )
```

### Parameters

`$settings`

Parameter type is `PD_Start_Source_Settings`. Available fields for this type are:

| Field Names | Default Values | Description |
|---|---|---|
| Timeout | PD_DEFAULT_TIMEOUT_INFINIT | Indicates the timeout (micro seconds) for connecting as SOURCE.<br>Should be greater than 5000us. |
| WithRa | PD_FALSE | Indicates whether to provide Ra on one of CC lines or not. |
| WithAccessory | PD_FALSE | If set to PD_TRUE, the command checks whether the UUT transitions to the AudioAccessory state or not. |
| AdvertizedCurrent | CC_RP_CUR_1_5 | Indicates advertised current level on Rp. |
| WithVConn | PD_FALSE | Indicates whether to turn on VConn or not. |
| VConnVoltage_mV | VCONN_VOLTAGE_DEFAULT | Indicates the VConn voltage level when start sourcing. |
| AccessoryStateDuration | 1000000 (us) | Time duration to check whether the UUT is in AudioAccessory state or not. |

### Result

None

### Examples

```
$startsrc_setting = PD_Start_Source_Settings
{
   WithRa = PD_TRUE
}
Call PD_SetStartSourceSetting( $startsrc_setting )
```

## 3.25 PD_StartSource

It starts SOURCE state machine for connecting to a Type-C device. The command is terminated if timeout occurs or the Exerciser transitions to Attached.SRC state. Usually is being called after `PD_SetStartSourceSetting` function.

### Format

```
Call PD_StartSource()
```

### Parameters

None

### Result

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

| Result Values |
|---|
| PD_RESULT_OK |
| PD_RESULT_FAILED |

### Examples

```
Call PD_StartSource()
```

## 3.26 PD_SetStartSinkSetting

It is used to customize behavior and characteristics of the Exerciser when acts as a SINK device. Settings will be applied to `PD_StartSink` command.

### Format

```
Call PD_SetStartSinkSetting( PD_Start_Sink_Settings $settings )
```

### Parameters

`$settings`

Parameter type is `PD_Start_Sink_Settings`. Available fields of this type are:

| Field Name | Default Values | Description |
|---|---|---|
| Timeout | PD_DEFAULT_TIMEOUT_INFINIT | Indicates the timeout (micro second) for connecting as a SINK. Should be greater than 5000us. |
| WithRa | PD_FALSE | Indicates whether to provide Ra on one of CC lines or not. |
| WithAccessory | PD_FALSE | Indicates whether to support SINKAS state machine or not. |
| AdvertizedCurrent | CC_RP_CUR_1_5 | When WithAccessory setting is PD_TRUE: indicates advertised current level on Rp. |
| StartWithSNK | PD_TRUE | Applies when WithAccessory setting is PD_TRUE. If is set to PD_FALSE, SINKAS state machine starts from Unattached.Accessory state instead of Unattached.SNK state. |
| AccessoryStateDuration | 1000000 us | When WithAccessory setting is PD_TRUE: indicates the time that Execiser stays in Powered.Accessory or Audio.Accessory states. |
| PoweredAccessoryExitState | PD_TYPE_C_STATE_NONE | When WithAccessory setting is PD_TRUE: indicates the exit state from Powered.Accessory state. |

### Result

None

### Examples

```
$startsnk_setting = PD_Start_Sink_Settings
{
   WithAccessory = PD_TRUE
}
Call PD_SetStartSinkSetting( $startsnk_setting )
```

## 3.27 PD_StartSink

It starts SINK or SINKAS state machine for connecting to a Type-C device. The command is terminated if timeout occurs or Exerciser transitions to Attached.SNK. When Exerciser acts as SINKAS, with no exit state for Powered.Accessory state, that state will be the last state and command is terminated after specified time for this state duration. Usually is being called after the `PD_SetStartSinkSetting` function.

### Format

```
Call PD_StartSink()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Values |
|---|
| PD_RESULT_OK |
| PD_RESULT_FAILED |

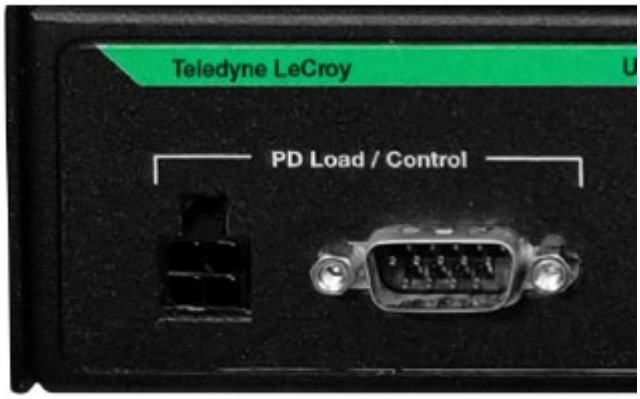
**Examples**

```
Call PD_StartSink()
```

## 3.28 Electronic Load box control

Voyager M310e platform supports a new capability to control external electronic load box through RS-232 interface. This allows PD exerciser directly control load box instead of legacy approach [in M310C/M310P/M4x that need to control through PC with API using external trigger cable]. This allows more flexibility with easier communication interface and simpler commands in PD script.

The new PD load interface connectors are located in rear of Voyager M310e.




### 3.28.1 PD_ConfigureLoadBox

It configures the Load Box connecting to the Analyzer board through RS-232 port.

**Format**

```
Call PD_ConfigureLoadBox( command, level_a, level_b )
```

**Parameters**

`command:`

| Possible Values | Description |
|---|---|
| PD_LOAD_BOX_RESET | |
| PD_LOAD_BOX_PRESET | |
| PD_LOAD_BOX_IMMEDIATE | |
| PD_LOAD_BOX_UPDATE | |
| PD_LOAD_BOX_TRIGGER | |
| PD_LOAD_BOX_DISCONNECT | |

```
level_a:
level_b:
   The current levels in milli ampear. The Load Box switches from level_a to level_b when
called with PD_LOAD_BOX_UPDATE, PD_LOAD_BOX_PRESET or PD_LOAD_BOX_IMMEDIATE.
```

**Result**

None

**Examples**

```
Call PD_ConfigureLoadBox( PD_LOAD_BOX_UPDATE, 1500, 1700 )
Call PD_ConfigureLoadBox( PD_LOAD_BOX_TRIGGER )
```

### 3.28.2 PD_WaitForLoadBox

It waits for the Load Box to be ready for work for the specified time.

**Format**

```
Call PD_WaitForLoadBox( timeout_us )
```

**Parameters**

`Timeout_us:`

The command timeout in micro second. If the Load Box is not ready and timeout occurs, the $PDResult.Result will be set to false to be used in `IfMatched/ElseMatched` command to make a decision.

**Result**

None

**Examples**

```
Call PD_WaitForLoadBox( 1000 ) # Wait for Load Box to be ready up to 1000 us
```

# 4 Basic Commands

## 4.1 PD_SendPacket

Sends the data payload towards the device. You can customize its behavior using provided settings.

### Format

```
Call PD_SendPacket(PD_Packet $send_packet, PD_SendPacketSettings $settings)
```

### Parameters

`$send_packet`

Defines the payload. Refer to `Packet Templates` for available packet templates.

`$settings`

Settings for sending packet. It should be inherited from `PD_SendPacketSettings` template.

Table below shows `PD_SendPacketSettings` structure in detail:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| OrderedSetType | PD_ORDERED_SET_TYPE_SOP(default)<br>PD_ORDERED_SET_TYPE_SOP_PRIME<br>PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME<br>PD_ORDERED_SET_TYPE_HARDRESET<br>PD_ORDERED_SET_TYPE_CABLERESET | Defines Ordered set type. |
| WaitForGoodCrc | PD_TRUE(default)<br>PD_FALSE | If the command should wait for peer GoodCrc message. |
| ResetOnError | PD_TRUE(default)<br>PD_FALSE | Send Soft Reset if relative GoodCrc has not been received, in case of sending SoftReset failure, HardReset will be sent. |
| RetryCount | PD_DEFAULT_RETRY_COUNT_REV_2(default.Rev2.0)<br>PD_DEFAULT_RETRY_COUNT_REV_3(default.Rev3.0) | Indicates the Retry Count. |
| RetryDelayTime | 0(default) | Delay time between two consecutive retries. |
| AutoMessageId | PD_TRUE(default)<br>PD_FALSE | To set MessageId automatically. |

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |
| PD_SUBRESULT_NO_GOODCRC | Subresult - No GoodCRC received for sent packet |
| PD_SUBRESULT_HARDRESET | Subresult - HardReset occurred. |
| PD_SUBRESULT_SOFTRESET | Subresult - SoftReset occurred. |

### Examples

```
#send a discover identity command
################################
$send_setting = PD_SendPacketSettings
{
   # could be PD_ORDERED_SET_TYPE_SOP_PRIME for cables
   OrderedSetType = PD_ORDERED_SET_TYPE_SOP
}
$discover_identity = PD_VDM_Discover_Identity_Message
Call PD_SendPacket( $discover_identity, $send_setting )

# Send Request message
######################
$request_data = PD_RequestDataObject_Fixed_Variable_NoGiveBack
{
    MaxOperatingCurrent_10mAUnits = 90
    OperatingCurrent_10mAUnits = 90
```

```
}
$request_packet = PD_RequestPacket
{
    Data = $request_data
}
#calling PD_SendPacket() command using default settings
$send_packet_settings = PD_SendPacketSettings
Call PD_SendPacket($request_packet, $send_packet_settings)
```

## 4.2 PD_SendPacket_Cable

Sends a packet as a Marked Cable towards the device.

### Format

```
Call PD_SendPacket_Cable( PD_Packet $send_packet,
                  PD_SendPacketSettings_Cable $settings )
```

### Parameters

`$send_packet`

Defines the payload. Refer to `Packet Templates` for available packet templates.

`$settings`

Settings for sending packet. It should be derived from `PD_SendPacketSettings_Cable` template.

`PD_SendPacketSettings_Cable` is derived from `PD_SendPacketSettings` template. Default values for some fields is changed as below:

```
OrderedSetType = PD_ORDERED_SET_TYPE_SOP_PRIME
ResetOnErrors = PD_FALSE
RetryCount = 0
```

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of possible result values:

| Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |
| PD_SUBRESULT_NO_GOODCRC | Subresult - No GoodCRC received for sent packet |

### Examples

```
#send a discover identity response
###################################
$send_setting = PD_SendPacketSettings_Cable

$header_vdo = PD_VDM_Discover_Identity_ID_Header_VDO
$stat_vdo = PD_VDM_Discover_Identity_Cert_Stat_VDO
$product_vdo = PD_VDM_Discover_Identity_Product_VDO
$cable_vdo = PD_VDM_Discover_Identity_Cable_VDO

$discover_identity_response = PD_VDM_Discover_Identity_Response
{
   VDOs = $header_vdo + $stat_vdo + $product_vdo + $cable_vdo
}
Call PD_SendPacket_Cable( $discover_identity_response, $send_setting )
```

## 4.3 Pd_SendErroneousChunks

Applicable to PD Rev 3.0 only. Sends packet chunks towards the UUT and has the ability to skip sending some packet chunks. User may set the `$PdGlobalSettings.UnchunkedSupport` (refer to `PD_Set`) setting to `PD_FALSE` before calling this function.

### Format

```
Pd_SendErroneousChunks( Pd_Packet $SentPacket, PD_SendErrChunksSettings
$SendErrChunksSettings )
```

### Parameters

`$send_packet`

Defines the payload. Refer to `Packet Templates` for available packet templates.

`$SendErrChunksSettings`

It should be inherited from `PD_SendErrChunksSettings` template. It includes all the fields of `PD_SendPacketSettings` (refer to `PD_SendPacket`) packet template but the default value for `RetryCount` field is `PD_DEFAULT_RETRY_COUNT_REV_3`. Following are additional fields of `PD_SendErrChunksSettings` packet template:

| Field Name | Default Values | Description |
|---|---|---|
| SkipSendChunkIndex | PD_INVALID_VALUE | Indicates from which chunk number the function should stop sending chunks. |

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |
| PD_SUBRESULT_NO_GOODCRC | Subresult - No GoodCRC received for sent packet |
| PD_SUBRESULT_HARDRESET | Subresult - HardReset occurred. |
| PD_SUBRESULT_SOFTRESET | Subresult - SoftReset occurred. |

### Examples

```
Packet TempExtendedPacket : Pd_Packet
{
    : Pd_MessageHeader
    : Pd_ExtendedMsgHeader
    : Pd_GenericPacket

    Extended = 1
}

Main
{
    .
    .
    .

    Pd_Set $PdGlobalSettings.UnchunkedSupport = PD_FALSE

    local $ErrChunkSettings = PD_SendErrChunksSettings
    {
        SkipSendChunkIndex = 4
    }

    local $SentPacket = TempExtendedPacket
    {
        MessageType = _00011111
        DataSize    = 260

        Data = { 00 01 02 03 }
    }

    call Pd_SendErroneousChunks($SentPacket, $ErrChunkSettings)
```

```
        .
        .
        .
}
```

## 4.4 PD_SendCorruptedPacket

Sends a packet towards the Unit Under Test which is corrupted intentionally.

### Format

```
Call PD_SendCorruptedPacket( PD_Packet $send_payload,    PD_SendCorruptedPacketSettings
$send_settings )
```

### Parameters

`$send_payload`

The payload to be sent. Refer to `Packet Templates` for available packet templates.

`$send_settings`

Settings for sending the corrupted payload. Setting type is `PD_SendCorruptedPacketSettings`:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| PreambleBitLen | 0x40(default) | Indicates the length of Preamble in bit. |
| NoPreamble | PD_TRUE<br>PD_FALSE(default) | Indicates whether to insert the Preamble or not. |
| OrderedsetType | PD_ORDERED_SET_TYPE_SOP(default),<br>PD_ORDERED_SET_TYPE_SOP_PRIME,<br>PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME,<br>PD_ORDERED_SET_TYPE_SOP_PRIME_DEBUG,<br>PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME_DEBUG<br>,<br>PD_ORDERED_SET_TYPE_HARDRESET,<br>PD_ORDERED_SET_TYPE_CABLERESET,<br>PD_ORDERED_SET_TYPE_INVALID | Indicates the ordered-set type. |
| CorruptedOrderedset | PD_TRUE<br>PD_FALSE(default) | If OrderedsetType field is PD_ORDERED_SET_TYPE_INVALID then content of this field will be replaced with ordered-set in the sent packet. |
| NoCrc | PD_TRUE<br>PD_FALSE(default) | Indicates whether to insert Crc in the packet or not. |
| CorruptCrc4b | PD_TRUE<br>PD_FALSE(default) | Indicates whether to corrupt Crc before 5-bit encoding or not. |
| CorruptCrc5b | PD_TRUE<br>PD_FALSE(default) | Indicates whether to corrupt Crc after 5-bit encoding or not. |
| CorruptCrc4bBitOffset | 0x00(default) | Indicates the bit offset (starting from 0) of 4-bit encoded Crc data to be corrupted. This field will be processed if CorruptCrc4b is PD_TRUE. |
| CorruptCrc4bBitLen | 0x00(default) | Indicates the length of corrupted Crc value which will be injected into Crc before 4-bit encoding. Will be processed if CorruptCrc4b is PD_TRUE. |
| CorruptCrc5bBitOffset | 0x00(default) | Indicates the bit offset (starting from 0) of 5-bit encoded Crc data to be corrupted. This field will be processed if CorruptCrc5b is PD_TRUE. |
| CorruptCrc5bBitLen | 0x00(default) | Indicates the length of corrupted Crc value which will be injected into Crc after 5-bit encoding. Will be processed if CorruptCrc5b is PD_TRUE. |
| CorruptCrc4bValue | 0x00(default) | Byte stream. Indicates the value to be replaced with Crc before 4-bit encoding value from CorruptCrc4bBitOffset with length of CorruptCrc4bBitLen. Will be processed if CorruptCrc4b is PD_TRUE. |
| CorruptCrc5bValue | 0x00(default) | Byte stream. Indicates the value to be replaced with Crc after 5-bit encoding value from |

| | | |
|---|---|---|
| | | CorruptCrc5bBitOffset with length of CorruptCrc5bBitLen. Will be processed if CorruptCrc5b is PD_TRUE. |
| CorruptPayload4b | PD_TRUE<br>PD_FALSE(default) | Indicates whether to corrupt Payload before 5-bit encoding or not. |
| CorruptPayload5b | PD_TRUE<br>PD_FALSE(default) | Indicates whether to corrupt Payload after 5-bit encoding or not. |
| CorruptPayload4bBitOffset | 0x00(default) | Indicates the bit offset of Payload (before 5-bit encoding) to get as the first data bit being corrupted (e.g. bit offset 0x08 means: get the Payload data corrupted starting from offset 0x08). This field will be processed if CorruptPayload4b is PD_TRUE. |
| CorruptPaylaod4bBitLen | 0x00(default) | Indicates the bit length of Payload (before 5-bit encoding) to get corrupted.(e.g. bit length 0x03 means: corrupt Payload( before 5-bit encoding) starting from CorruptPayload4bBitOffset and length of 0x03 bits). This field will be processed if CorruptPayload4b is PD_TRUE. |
| CorruptPaylaod4bValue | 0x00(default) | Byte stream. Defines the value to be replaced with the Payload (before 5-bit encoding) data. The offset and length of replacing data should be defined using CorruptPayload4bBitOffset and CorruptPaylaod4bBitLen fields. This field will be processed if CorruptPayload4b field is PD_TRUE. |
| CorruptPyload5bBitOffset | 0x00(default) | Indicates the bit offset of Payload (after 5-bit encoding) to get as the first data bit being corrupted (e.g. bit offset 0x08 means: get the Payload data corrupted starting from offset 0x08).<br>This field will be processed if CorruptPayload5b field is PD_TRUE. |
| CorruptPyload5bBitLen | 0x00(default) | Indicates the bit length of Payload (after 5-bit encoding) to get corrupted.(e.g. bit length 0x03 means: corrupt Payload( after 5-bit encoding) starting from CorruptPyload5bBitOffset and length of 0x03 bits).This field will be processed if CorruptPayload5b field is PD_TRUE. |
| CorruptPyload5bValue | 0x00(default) | Byte stream. Defines the value to be replaced with the Payload (after 5-bit encoding) data. The offset and length of replacing data should be defined using CorruptPyload5bBitOffset and CorruptPyload5bBitLen fields. Will be processed if CorruptPayload5b field is PD_TRUE. |
| NoEop | PD_TRUE<br>PD_FALSE(default) | Indicates whether to insert EOP in the packet or not. |
| CorruptEop | PD_TRUE<br>PD_FALSE(default) | Indicates whether to corrupt EOP in th packet or not |
| CorruptedEopSymbol | 0x00(default) | COrrupted EOP symbol to be replaced with EOP in the packet. This field will be processed if CorruptEop field is PD_TRUE. |
| OperationType | PD_CORRUPT_OPERATION_TYPE_CUSTOM_VALUE(default)<br>PD_CORRUPT_OPERATION_TYPE_FLIP_ALL_BITS<br>PD_CORRUPT_OPERATION_TYPE_INCREMENT_MSG_ID | Defines the operation type which will be applied. Only one operation type can be chosen at a time.<br><br>Default operation which is PD_CORRUPT_OPERATION_TYPE_CUSTOM_VALUE will be replacing data from the specified offset for specified length.<br><br>PD_CORRUPT_OPERATION_TYPE_FLIP_ALL_BITS will be flipping all the bits in between the specified offset to the specified length.<br><br>PD_CORRUPT_OPERATION_TYPE_INCREMENT_MSG_ID<br>will be incrementing MsgID of the $send_message |

## Result

None

### Examples

```
$GetSinkCapsPacket = PD_GetSinkCapMessage
{
    PortPowerRole_CablePlug = 1
}
$corrupted_send_settings = PD_SendCorruptedPacketSettings
{
    CorruptCrc4b = PD_TRUE
}

call PD_SendCorruptedPacket($GetSinkCapsPacket, $corrupted_send_settings)
```

## 4.5 PD_ReceivePacket

Receives a packet from device. You can specify the packet type using its settings.

### Format

```
Call PD_ReceivePacket( PD_ReceivePacketSettings $receive_Settings )
```

### Parameters

`$receive_Settings`

Settings for receiving packet. The structure type should be `PD_ReceivePacketSettings`.

Table below shows this structure in detail:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| OrderedSetType | PD_ORDERED_SET_TYPE_SOP(default)<br>PD_ORDERED_SET_TYPE_SOP_PRIME<br>PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME<br>PD_ORDERED_SET_TYPE_HARDRESET<br>PD_ORDERED_SET_TYPE_CABLERESET | Ordered set type for receiving message. |
| PacketType | PD_MESSAGE_TYPE_ANY(default)<br>PD_MESSAGE_TYPE_GOODCRC<br>PD_MESSAGE_TYPE_GOTO_MIN<br>PD_MESSAGE_TYPE_ACCEPT<br>PD_MESSAGE_TYPE_REJECT<br>PD_MESSAGE_TYPE_PING<br>PD_MESSAGE_TYPE_PS_RDY<br>PD_MESSAGE_TYPE_GET_SOURCE_CAP<br>PD_MESSAGE_TYPE_GET_SINK_CAP<br>PD_MESSAGE_TYPE_DR_SWAP<br>PD_MESSAGE_TYPE_PR_SWAP<br>PD_MESSAGE_TYPE_VCONN_SWAP<br>PD_MESSAGE_TYPE_WAIT<br>PD_MESSAGE_TYPE_SOFT_RESET<br>PD_MESSAGE_TYPE_HARD_RESET<br>PD_MESSAGE_TYPE_CABLE_RESET<br>PD_MESSAGE_TYPE_NOT_SUPPORTED<br>PD_MESSAGE_TYPE_GET_SRC_CAP_EXT<br>PD_MESSAGE_TYPE_GET_STATUS<br>PD_MESSAGE_TYPE_FR_SWAP<br>PD_MESSAGE_TYPE_GET_PPS_STATUS<br>PD_MESSAGE_TYPE_GET_COUNTRY_CODES<br>PD_MESSAGE_TYPE_GET_SNK_CAP_EXT<br>PD_MESSAGE_TYPE_SOURCE_CAP<br>PD_MESSAGE_TYPE_REQUEST<br>PD_MESSAGE_TYPE_BIST<br>PD_MESSAGE_TYPE_SINK_CAP<br>PD_MESSAGE_TYPE_BATTERY_STATUS<br>PD_MESSAGE_TYPE_ALERT<br>PD_MESSAGE_TYPE_GET_COUNTRY_INFO<br>PD_MESSAGE_TYPE_VDM<br>PD_MESSAGE_TYPE_SRC_CAP_EXT<br>PD_MESSAGE_TYPE_STATUS<br>PD_MESSAGE_TYPE_GET_BATTERY_CAP<br>PD_MESSAGE_TYPE_GET_BATTERY_STATUS<br>PD_MESSAGE_TYPE_BATTERY_CAP<br>PD_MESSAGE_TYPE_GET_MANUFACTURER_INFO<br>PD_MESSAGE_TYPE_MANUFACTURER_INFO<br>PD_MESSAGE_TYPE_SECURITY_REQUEST<br>PD_MESSAGE_TYPE_SECURITY_RESPONSE<br>PD_MESSAGE_TYPE_PPS_STATUS<br>PD_MESSAGE_TYPE_COUNTRY_INFO | Message type to receive. |

| | | |
|---|---|---|
| | PD_MESSAGE_TYPE_COUNTRY_CODES<br>PD_MESSAGE_TYPE_SNK_CAP_EXT | |
| VdmCommand | PD_VDM_COMMAND_ANY(default)<br>PD_VDM_COMMAND_DISCOVER_IDENTITY<br>PD_VDM_COMMAND_DISCOVER_SVIDS<br>PD_VDM_COMMAND_DISCOVER_MODES<br>PD_VDM_COMMAND_ENTER_MODE<br>PD_VDM_COMMAND_EXIT_MODE<br>PD_VDM_COMMAND_DISPLAYPORT_STATUS_UPDATE<br>PD_VDM_COMMAND_DISPLAYPORT_CONFIGURE<br>PD_VDM_COMMAND_ATTENTION | VDM command. |
| VdmCommandType | PD_VDM_COMMAND_TYPE_INITIATOR(default)<br>PD_VDM_COMMAND_TYPE_RESPONDER_ACK<br>PD_VDM_COMMAND_TYPE_RESPONDER_NAK<br>PD_VDM_COMMAND_TYPE_RESPONDER_BUSY<br>PD_VDM_COMMAND_TYPE_ANY | VDM command type. |
| AutoGoodCrc | PD_TRUE(default)<br>PD_FALSE | Send GoodCrc on receiving a message, automatically. |
| DelayBeforeGoodCrc | 0(default)<br>Or other user defined value. | Delay before sending GoodCrc message. |
| WaitTimeOut | PD_DEFAULT_TIMEOUT_SENDER_RESPONSE(default)<br>PD_DEFAULT_TIMEOUT_INFINIT<br>Or other user defined value. | Receive timeout(micro second). |
| DiscardPrevReceived | PD_TRUE<br>PD_FALSE(default) | Discards any (unprocessed) packet received before calling PD_ReceivePacket function. |
| ReturnOnUnexpectedPkt | PD_TRUE<br>PD_FALSE(default) | If set to PD_TRUE, cause PD_ReceivePacket() function to return on receiving unexpected packet. |
| NoSoftResetResponse | PD_TRUE<br>PD_FALSE(default) | If set to PD_TRUE, the PD_ReceivePacket() function will not respond to any received Soft_Resets. |
| NoHardResetResponse | PD_TRUE<br>PD_FALSE(default) | If set to PD_TRUE, the PD_ReceivePacket() function will not respond to any received Hard_Resets. |

## Result

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |
| PD_SUBRESULT_RECEIVE_TIMEOUT | Subresult - No packet received within specified time |
| PD_SUBRESULT_UNEXPECTED_MSG_RECEIVED | Subresult - Unexpected packet received |
| PD_SUBRESULT_HARDRESET | Subresult - HardReset received |
| PD_SUBRESULT_SOFTRESET | Subresult - SoftReset received |

## Examples

```
#Receive source caps
####################
# Wait to receive source capability. GoodCRC is sent automatically.
$recv_settings = PD_ReceivePacketSettings
{
    WaitTimeOut = PD_DEFAULT_TIMEOUT_INFINIT
    PacketType = PD_MESSAGE_TYPE_SOURCE_CAP
}
call PD_ReceivePacket($recv_settings)

#Receive VDM message
####################
$receive_settings = PD_ReceivePacketSettings
{
    PacketType = PD_MESSAGE_TYPE_VDM
}
call PD_ReceivePacket( $receive_settings )
```

## 4.6 PD_SendSoftReset

Sends Soft Reset and performs the reset according to the selected Ordered-Set Type.

**Format**

```
Call PD_SendSoftReset( orderedset_type )
```

**Parameters**

`orderedset_type`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

If sending SoftReset succeeded the result is `PD_RESULT_OK`. In case of failure it may lead to Power Negotiation.

**Examples**

```
Call PD_SendSoftReset( PD_ORDERED_SET_TYPE_SOP )
```

## 4.7 PD_SendHardReset

Sends Hard Reset and performs the reset.

**Format**

```
Call PD_SendHardReset()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_SendHardReset()
```

## 4.8 PD_SendCableReset

Sends Cable Reset and resets all the cable related states in protocol layer.

**Format**

```
Call PD_SendCableReset()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_SendCableReset()
```

## 4.9 PD_DelayNoAutoResponse

Delays Exerciser execution for specified time.

**Format**

```
Call PD_DelayNoAutoResponse( Micro_Sec )
```

**Parameters**

`Micro_Sec`

Delay in micro seconds.

**Result**

None

**Examples**

```
#calling PD_DelayNoAutoResponse
Call PD_DelayNoAutoResponse(15000)
```

## 4.10 PD_Delay

Delays Exerciser execution for specified time.

**Format**

```
Call PD_Delay( delay_value )
```

**Parameters**

`delay_value`

Delay in micro seconds.

**Result**

None

**Examples**

```
#calling PD_Delay
Call PD_Delay(15000)
```

## 4.11 PD_SetRoles

Sets data role and power role of Exerciser.

**Format**

```
Call PD_SetRoles( DataRole, PowerRole )
```

**Parameters**

`DataRole`

Possible values:

```
PD_PORT_DATA_ROLE_UFP
PD_PORT_DATA_ROLE_DFP
```

`PowerRole`

Possible values:

```
PD_PORT_POWER_ROLE_SINK
PD_PORT_POWER_ROLE_SOURCE
```

**Examples**

```
Call PD_SetRoles( PD_PORT_DATA_ROLE_DFP, PD_PORT_POWER_ROLE_SOURCE )
```

## 4.12 PD_Set

Using this command you can change necessary settings or variables inside the Exerciser.

**Format**

```
PD_Set $PdGlobalSettings.<field_name> = <value>
PD_Set $PdTimers.<field_name> = <value>
```

**Parameters**

List of `$PdGlobalSettings` fields:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| PortDataRole | PD_PORT_DATA_ROLE_DFP<br>PD_PORT_DATA_ROLE_UFP(default) | Defines port data role. |
| PortPowerRole | PD_PORT_POWER_ROLE_SINK(default)<br>PD_PORT_POWER_ROLE_SOURCE | Defines port power role. |
| CheckMessageId | PD_FALSE(Default)<br>PD_TRUE | Enables/Disables received packet message id verification. |
| SpecificationRevision | PD_SPEC_REVISION_1<br>PD_SPEC_REVISION_2(Default)<br>PD_SPEC_REVISION_3<br>Or any user defined value. | Changes the SpecificationRevision of all messages sent by the Exerciser.<br><br>**Note 1 (Rev.3.0 or higher, all messages except non-SOP messages which are sent by PD Exerciser to a cable plug):** This setting will be applied only if the *AutoSpecRev* setting of PD_SetNegotiationSetting_Source (PD Exerciser operating as Source) or PD_SetNegotiationSetting_Sink (PD Exerciser operating as Sink) has been set to PD_FALSE.<br><br>**Note 2:** To set the Specification Revision of GoodCrc messages which are sent by PD Exerciser, refer to *AutoGoodCRCSpecRev* and *GoodCRCSpecRev* global settings. |
| EnableCableEmulator | PD_FALSE(default)<br>PD_TRUE | Enables/Disables SOP Cable Emulator engine in Exerciser. If enabled, the Exerciser simulates a SOP Prime Marked Cable as well as source or sink PD Device. *It should be set only once in the target Exerciser Script*. |

| | | |
|---|---|---|
| EnableCableDPrimeEmulator | PD_FALSE (default)<br>PD_TRUE | Enables/Disables SOP Double Prime Cable Emulator engine in Exerciser. If enabled, the Exerciser simulates a SOP Double Prime Marked Cable as well as source or sink PD Device. *It should be set only once in the target Exerciser Script*. |
| EnableDeviceEmulator | PD_FALSE,<br>PD_TRUE(default) | If disabled, Device Emulator AutoResponse will be disabled (in case of Exerciser acting as a Sink or Source device). |
| NegotiateAfterReset | PD_FALSE,<br>PD_TRUE (default) | If set to PD_TRUE, then the Exerciser will run Negotiation after receiving/sending SoftReset or HardReset. |
| VConnPassThrough | PD_FALSE(default),<br>PD_TRUE | Indicates whether the Exerciser is connected to the DUT using a VConn Pass Through cable or not. |
| PDWorkingRevision | PD_SPEC_REVISION_2(default)<br>PD_SPEC_REVISION_3 | Sets the Power Delivery working revision. *It should be set only once in the target Exerciser Script*. Its recommended to change this setting using PD_SetWorkingRevision high-level function. |
| UnchunkedSupport | PD_FALSE,<br>PD_TRUE (default) | Indicates whether to support sending un-chunked messages or not. |
| StructuredVDMVersion | PD_INVALID_VALUE(default)<br>Or any user defined value. | Indicates the VDM version of structured VDM messages. If the value is PD_INVALID_VALUE then the Exerciser assigns the proper value for structured VDM version according to current operational Power Delivery Revision. |
| StructuredVDMSVID | PD_VDM_SID(default)<br>Or any user defined value. | Indicates the SVID of structured VDM messages. |
| VConnThroughConnectCCLine | CC_LINE_1 (default)<br>CC_LINE_2 | Indicates the desired CC line which makes the PD Exerciser to try to establish the Type-C connection on this CC line. This setting is only applicable when the cable is a VConnPassThrough cable. |
| FRSwapSupport | PD_TRUE (default)<br>PD_FALSE | Applies only if the run-time working revision is Rev.3.0 or higher. Indicates whether the Fast Role Swap is supported by Pd Exerciser engine or not. |
| AutoGoodCRCSpecRev | PD_TRUE (default)<br>PD_FALSE | If set to PD_TRUE, proper Specification Revision will be set in each GoodCRC message sent to peer port (SOP or non-SOP) depending on the run-time working revision and whether the PD Exerciser is operating as Cable Emulator or not. Otherwise the value of $PdGlobalSettings.GoodCRCSpecRev will be used. |
| GoodCRCSpecRev | PD_SPEC_REVISION_1 (default)<br>PD_SPEC_REVISION_2<br>PD_SPEC_REVISION_3 | If $PdGlobalSettings.AutoGoodCRCSpecRev is PD_FALSE, then this value will be used as Specification revision of all GoodCRC messages sent by PD Exerciser. |
| InitiateAMS | PD_TRUE (default)<br>PD_FALSE | Applies If the run-time working revision is Rev.3.0 or higher. If set to PD_TRUE, then the PD Exerciser will always check for SinkTxNG and SinkTxOk values, regarding to its current role as Source or Sink, before starting an AMS. |
| SpecificationRevisionToCable | PD_SPEC_REVISION_1<br>PD_SPEC_REVISION_2<br>PD_SPEC_REVISION_3 (default) | Applies if the run-time working revision is Rev.3.0 or higher. If *AutoSpecRevCable* setting of PD_SetDiscoverIdentitySetting has been set to PD_FALSE, then this value will be used as Specification Revision of all non-SOP messages which are sent by PD Exerciser to cable plug. |
| DisableVBusOn | PD_TRUE<br>PD_FALSE (default) | If set to PD_TRUE, then PD Exerciser will not turn the VBus on. |
| CheckBatteryRef | PD_TRUE (default)<br>PD_FALSE | Applies if the run-time working revision is Rev.3.0 or higher. If set to PD_TRUE, then PD Exerciser will check the Battery Reference received by Get_Battery_Status, Alert, Get_Battery_Cap and Manufacturer_info messages with the number of batteries defined by PD Exerciser (using the PD_SetSrcCapExtDataBlock command) and replies with a proper message. |
| CheckCountryCode | PD_TRUE (default)<br>PD_FALSE | Applies if the run-time working revision is Rev.3.0 or higher. If set to PD_TRUE, then PD Exerciser will check the Country Codes received by Get_Country_info message with the available country codes which has been added using the Pd_SetCountryCodesDataBlock function and replies with a proper message. |

| | | |
|---|---|---|
| | | **Note:** PD Exerciser checks the received country codes with only the first 15 country codes which has been added by `Pd_SetCountryCodesDataBlock` function. |
| ChangeHWSettingDelay | 5000(default) | Indicates how much delay (in micro seconds) is needed to apply after changing any Hardware Setting (e.g. changing Rp/Rd/Ra resistors or inserting/removing a capacitor etc.). |
| SetVbusDelay | 20000(default) | Indicates how much delay (in micro seconds) is needed to apply after changing the VBus state. |
| AutoSinkCapsFromSourcCaps | PD_TRUE<br>PD_FALSE (default) | It is used when Exerciser is Sink and you don't want to add any Sink Caps manually. It causes all Source Caps to be added as Sink Caps Dynamically. |

List of `$PdTimers` fields(for detailed description refer to Power Delivery Specification):

| Field Name | Description |
|---|---|
| tTypeCSinkWaitCap | Default: 620000 us |
| tTypeCSendSourceCap | Default: 150000 us |
| tPSTransition | Default: 550000 us |
| tPSSourceOff | Default: 920000 us |
| tPSSourceOn | Default: 480000 us |
| tSrcTransition | Default: 25000 us |
| tDiscoverIdentity | Default: 45000 us |
| tSafe0V | Default: 650000 us |
| tSafe5V | Default: 275000 us |
| tPSHardResetMin | Default: 25000 us |
| tPSHardResetMax | Default: 35000 us |
| tPSHardReset | Default: 30000 us |
| tSrcRecoverMin | Default: 660000 us |
| tSrcRecoverMax | Default: 1000000 us |
| tSrcRecover | Default: 1000000 us |
| tReceive | Default: 1100 us |
| tVCONNStable | Default: 50000 us |
| tVCONNSourceOff | Default: 25000 us |
| tVCONNSourceOn | Default: 50000 us |
| tVCONNSourceTimeout | Default: 200000 us |
| tVCONNSourceTimeoutMin | Default: 100000 us |
| tVCONNSourceTimeoutMax | Default: 200000 us |
| tVCONNZero | Default: 120000 us |
| tSenderResponse | Default: 30000 us |
| tBISTContMode | Default: 60000 us |
| tVDMBusy | Default: 50000 us |
| tVDMWaitModeEntry | Default: 50000 us |
| tVDMWaitModeExit | Default: 50000 us |
| tSwapSourceStart | Default: 20000 us |
| tSrcSwapStdbyMax | Default: 650000 us |
| tNewSrcMax | Default: 275000 us |
| tDRP | Default: 80000 us |
| dcSRC_DRP | Default: 50(time percent) |
| tCCDebounce | Default: 100000 us |
| tCCDebounceMin | Default: 100000 us |
| tCCDebounceMax | Default: 200000 us |
| tDRPTry | Default: 75000 us |
| tDRPTryMin | Default: 75000 us |
| tDRPTryMax | Default: 150000 us |
| tDRPTryWait | Default: 400000 us |
| tDRPTryWaitMin | Default: 400000 us |
| tDRPTryWaitMax | Default: 800000 us |
| tPDDebounce | Default: 10000 us |
| tPDDebounceMin | Default: 10000 us |
| tPDDebounceMax | Default: 20000 us |
| tTryCCDebounce | Default: 20000 us |
| tTryTimeout | Default: 1100000 us |
| tSinkTx | Default: 18000 us |

| | |
|---|---|
| tFRSwapTx | Default: 110 us |
| tFRSwapInitMax | Default: 15000 us |
| tFRSwapInitMin | Default: 500 us |
| tFRSwapComplete | Default: 0 us |
| tFRSwap5V | Default: 0 us |
| tErrorRecovery | Default: 25000 us |
| tChunkSenderResponse | Default: 30000 us |
| tChunkingNotSupported | Default: 50000 us |

**Result**

None

**Examples**

```
# Enables cable emulator
PD_Set $PdGLOBALSETTINGS.EnableCableEmulator = PD_TRUE

Main
{
   # Sets GoodCRC timeout
   PD_Set $PdTimers.tReceive = 950

   Call PD_WaitForDiscoverIdentity_Cable()
}
```

## 4.13 IfMatched/ElseMatched

Compares Exerciser settings, Received Packet Fields and Command Results to a desired value.

Using this command you can compare Exerciser settings or variables to other Exerciser settings or variables or to a constant.

**Format**

```
Ifmatched(<1st_operand>, <2nd_operand>, <operator>)
{
   #command list
}
```

```
[
ElseMatched(<1st_operand>, <2nd_operand>, <operator>)
{
 #command list
}
 #more optional ElseMatched(<1st_operand>, <2nd_operand>, <operator>) here
   .
   .
   .
ElseMatched
{
     #command list
}
]
IfMatchedEnd

* ElseMatched clause is optional
```

## Parameters

### 1st_operand

1st operand should be in one of the following formats:

```
$PdGlobalSettings.<field_name>
$PdResult.<field_name>
$<packet_variable>.<field_name>
$CCLinesStatus.<field_name>
```

List of `$PdResult` fields:

| Field Name | Description |
|---|---|
| Result | Last executed command result |
| Subresult | Last executed command subresult (in case of failure, this field describes the reason) |
| LastReceivedPacketOrderedSet | Last received packet ordered set type |
| LastReceivedPacketType | Last received packet type |
| LastReceivedPacketPowerRole | Last received packet power role field value |
| LastReceivedPacketDataRole | Last received packet data role field value |
| LastReceivedPacketSentToCable | Indicates whether the last received packet has been sent to cable(packet towards the cable) or not |
| LastReceivedPacketMsgID | Last received packet `MessageId` field value |
| LastReceivedPacketVdmCommand | Last received packet VDM command value, if the packet is VDM packet |
| LastReceivedPacketVdmCommandType | Last received packet VDM command type value, if the packet is VDM packet |
| LastReceivedPacketVdmSVID | Last received packet SVID, if the packet is a VDM packet |
| LastReceivedPacketVdmObjPos | Last received packet `ObjetctPosition`, if the packet is a VDM packet |
| LastSelectedCapIndex | Last selected Sink/Source capability index during power negotiation. |
| LastRequestHasMismatch | Last received packet `HasMismatch` field value, if the packet is Request message |
| ExplicitContract | Indicates whether explicit contract is established or not. |
| LastReceivedUnchunkedSupport | Indicates whether the UUT supports Unchuncked messages or not. Applicable to PD Rev3.0 only. |
| DataRoleSwapped | Indicates whether the DataRole swapped or not comparing to the initial value of PD Exerciser DataRole. |
| PowerRoleSwapped | Indicates whether the PowerRole swapped or not comparing to the initial value of PD Exerciser PowerRole. |
| AgreedSVDMVersionMajor | The agreed common Structured VDM Major Version |
| AgreedSVDMVersionMinor | The agreed common Structured VDM Minor Version |

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

List of `$CCLinesStatus` fields:

| Field Name | Description |
|---|---|
| CC1_UUT_Float | |
| CC1_UUT_Rp | |
| CC1_UUT_Rd | |
| CC1_UUT_Ra | |
| CC1_UUT_Vconn | |
| CC1_UUT_Rp_Cur | |
| CC1_Exr_Rd | |
| CC1_Exr_Ra | |
| CC1_Exr_Rp_Cur | |
| CC1_Exr_Vconn | |
| CC2_UUT_Float | |
| CC2_UUT_Rp | |
| CC2_UUT_Rd | |
| CC2_UUT_Ra | |
| CC2_UUT_Vconn | |
| CC2_UUT_Rp_Cur | |
| CC2_Exr_Rd | |
| CC2_Exr_Ra | |
| CC2_Exr_Rp_Cur | |
| CC2_Exr_Vconn | |
| VbusVSafe0 | |
| VbusVSafe5 | |

For available `$PdGlobalSettings` fields refer to **PD_Set**.

**2nd_operand**

It could be as `<1st_operand>` or a constant `<value>`.

**operator**

List of possible values for operator:

```
PD_COMPARE_EQUAL
PD_COMPARE_GREATER
PD_COMPARE_LESS
PD_COMPARE_NOT_EQUAL
```

## Result

None

## Examples

```
$send_setting = PD_SendPacketSettings
{
   ResetOnError = PD_FALSE
    OrderedSetType = PD_ORDERED_SET_TYPE_SOP
}
$receive_settings = PD_ReceivePacketSettings
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

```
{
    PacketType = PD_MESSAGE_TYPE_VDM
}
#send the packet
$discover_identity = PD_VDM_Discover_Identity_Message
Call PD_SendPacket( $discover_identity, $send_setting )

#check for result
IfMatched( $PdResult.Result, PD_RESULT_OK, PD_COMPARE_EQUAL )
{
   Call PD_ReceivePacket( $receive_settings )
}
ElseMatched( $PdResult.Result, PD_RESULT_FAILED, PD_COMPARE_EQUAL )
{
   Call PD_SendHardReset()
}
ElseMatched
{
   $ping_msg = PD_PingMessage
   Call PD_SendPacket( $ping_msg, $send_setting )
}
IfMatchedEnd
```

## 4.14 PD_Loop

Using this command you can create a loop containing other Exerciser commands.

**Note** - The limit for using nested `PD_Loop()` commands is 8.

### Format

```
PD_Loop(count)
{
    #command list
}
```

### Parameters

`count`

Loop count

### Result

None

### Examples

```
$send_setting = PD_SendPacketSettings
{
    OrderedSetType = PD_ORDERED_SET_TYPE_SOP
}
$ping_msg = PD_PingMessage

PD_Loop(3)
{
   call PD_SendPacket( $ping_msg, $send_setting )
}
```

## 4.15 PD_TimerLoop

Using this command you can create a loop(containing other Exerciser commands) which is bound to a predefined timer. On timer timeout, the loop will exit.

**Note** - The limit for using nested `PD_TimerLoop()` commands is 8.

### Format

```
PD_TimerLoop(timeout)
{
    #command list
```

```
}
```

**Parameters**

`timeout`

Loop duration in Micro Seconds.

**Result**

None

**Examples**

```
$send_setting = PD_SendPacketSettings
{
    OrderedSetType = PD_ORDERED_SET_TYPE_SOP
}
$ping_msg = PD_PingMessage

# Sending Ping message for 200ms
PD_TimerLoop(200000)
{
   call PD_SendPacket( $ping_msg, $send_setting )
}
```

## 4.16 PD_Stop

Stops the Exerciser.

**Format**

```
Call PD_Stop( return_value )
```

**Parameters**

`return_value`

Value returned to Exerciser. The possible values are 1 to 255.

**Result**

None

**Examples**

```
Call PD_Stop( 100 )
```

## 4.17 PD_Disconnect

Simulates cable detach.

**Format**

```
Call PD_Disconnect()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_Disconnect()
```

## 4.18 PD_StartUSBExerciser

Starts specified Exercisers which are waiting for PD Exerciser's permission.

**Format**

```
Call PD_StartUSBExerciser( ExercisersTypes )
```

**Parameters**

`ExercisersTypes`

Possible values:

```
PD_USB2_EXERCISER
PD_USB3_EXERCISER
PD_USB4_EXERCISER
PD_TBT3_EXERCISER
```

**Result**

None

**Examples**

```
Call PD_StartUSBExerciser( PD_USB2_EXERCISER | PD_USB3_EXERCISER )
```

## 4.19 PD_RunUSB3TermDetection

Directs the Exerciser to actively perform USB 3 Rx Termination detection. A TERM State (Report) event will be displayed in the trace.

**NOTE:** This command is state agnostic. Performing Rx Termination detection in certain states could interfere with SS communication and result in unexpected link behavior.

**Format**

```
Call PD_RunUSB3TermDetection()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_RunUSB3TermDetection()
```

## 4.20 PD_ResumeUSB2Exerciser

Resumes USB2 Exerciser execution paused by Break command.

**Format**

```
Call PD_ResumeUSB2Exerciser()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResumeUSB2Exerciser()
```

## 4.21 PD_ReportUSB3TermStatus

Reports the last-known USB 3 Rx Termination status, typically from when the Exerciser was most recently in eSS.Inactive, Rx.Detect, U2, or U3. A TERM State (Report) event will be displayed in the trace.

**Format**

```
Call PD_ReportUSB3TermStatus()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ReportUSB3TermStatus()
```

## 4.22 PD_IncreaseMsgId

Increase Message ID(Exerciser mode: DFP/UFP).

**Format**

```
Call PD_IncreaseMsgId(OrderedSetType)
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
Call PD_IncreaseMsgId(PD_ORDERED_SET_TYPE_SOP)
```

## 4.23 PD_DecreaseMsgId

Decrease Message ID(Exerciser mode: DFP/UFP).

**Format**

```
Call PD_DecreaseMsgId(OrderedSetType)
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
Call PD_DecreaseMsgId(PD_ORDERED_SET_TYPE_SOP)
```

## 4.24 PD_IncreaseMsgId_Cable

Increase Message ID(Exerciser mode: Cable Emulator).

**Format**

```
Call PD_IncreaseMsgId_Cable(OrderedSetType)
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
Call PD_IncreaseMsgId_Cable(PD_ORDERED_SET_TYPE_SOP_PRIME)
```

## 4.25 PD_DecreaseMsgId_Cable

Decrease Message ID(Exerciser mode: Cable Emulator).

**Format**

```
Call PD_DecreaseMsgId_Cable(OrderedSetType)
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
Call PD_DecreaseMsgId_Cable(PD_ORDERED_SET_TYPE_SOP_PRIME)
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

## 4.26 PD_SendExternalTriggerOut

Sends the external trigger out.

### Format

```
Call PD_SendExternalTriggerOut()
```

### Parameters

None

### Result

None

### Examples

```
Call PD_SendExternalTriggerOut()
```

## 4.27 PD_InterruptAMS

Sends a packet to interrupt an AMS. This command waits to receive a packet of an AMS then immediately after sending the GoodCRC, the interrupting packet is sent. You can customize its behavior using the provided settings.

### Format

```
Call PD_InterruptAMS( Pd_Packet $interrupting_packet, PD_InterruptAMSSettings $settings )
```

### Parameters

`$interrupting_packet`

Refer to `Packet Templates` for available packet templates.

`$settings`

Settings for sending the interrupting packet. It should be inherited from `PD_InterruptAMSSettings` template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| InterruptingPktOS | PD_ORDERED_SET_TYPE_SOP (default)<br>PD_ORDERED_SET_TYPE_SOP_PRIME<br>PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME<br>PD_ORDERED_SET_TYPE_HARDRESET<br>PD_ORDERED_SET_TYPE_CABLERESET | Defines Ordered set type for the interrupting packet. |
| InterruptingPktAutoMessageId | PD_TRUE (default)<br>PD_FALSE | To automatically set MessageId for the interrupting packet. |
| InterruptingPktResponseNeedsGoodCRC | PD_FALSE (default)<br>PD_TRUE | Indicates whether to send a GoodCRC to the response of the interrupting packet or not. |
| InterruptingPktDelay | 0 (default) | Indicates the delay (in micro-seconds) before sending the interrupting packet |
| FirstReceivePktTimeout | PD_DEFAULT_TIMEOUT_SENDER_RESPONSE (default)<br>PD_DEFAULT_TIMEOUT_INFINIT<br>Or other user defined value. | timeout for receiving the first packet (in microsecond). |
| FirstReceivedPktIsGoodCRC | PD_FALSE (default)<br>PD_TRUE | Indicates whether the first received packet is a GoodCRC message or not. |
| FirstReceivedPktNeedsGoodCRC | PD_TRUE (default)<br>PD_FALSE | Indicates whether to send a GoodCRC to the first received message or not. |
| GoodCRCCorruptCrc4b | PD_FALSE (default) | It is used to corrupt CRC32 of the GoodCRC packet of the first received |

| | | |
|---|---|---|
| | PD_TRUE | packet before 5-bit encoding. It will be flipping all the bits specified by GoodCRCCorruptCrc4bBitOffset and GoodCRCCorruptCrc4bBitLen. This field will be processed if FirstReceivedPktNeedsGoodCRC is PD_TRUE. |
| GoodCRCCorruptCrc4bBitOffset | 0x00(default) | Indicates the bit offset (starting from 0) of 4-bit encoded CRC32 data to be corrupted. This field will be processed if GoodCRCCorruptCrc4b is PD_TRUE. |
| GoodCRCCorruptCrc4bBitLen | 0x00(default) | Indicates the length of corrupted CRC32 value which will be injected into CRC32 before 4-bit encoding. Will be processed if GoodCRCCorruptCrc4b is PD_TRUE. |
| GoodCRCCorruptCrc5b | PD_FALSE (default)<br>PD_TRUE | It is used to corrupt CRC32 of the GoodCRC packet of the first received packet after 5-bit encoding. It will be flipping all the bits specified by GoodCRCCorruptCrc5bBitOffset and GoodCRCCorruptCrc5bBitLen. This field will be processed if FirstReceivedPktNeedsGoodCRC is PD_TRUE. |
| GoodCRCCorruptCrc5bBitOffset | 0x00(default) | Indicates the bit offset (starting from 0) of 5-bit encoded CRC32 data to be corrupted. This field will be processed if GoodCRCCorruptCrc5b is PD_TRUE. |
| GoodCRCCorruptCrc5bBitLen | 0x00(default) | Indicates the length of corrupted CRC32 value which will be injected into CRC32 after 5-bit encoding. Will be processed if GoodCRCCorruptCrc5b is PD_TRUE. |

## Result

User can evaluate the command results (including sub-results) using IfMatched/ElseMatched command.

| Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |
| PD_SUBRESULT_RECEIVE_TIMEOUT | Subresult - No packet received within specified time |
| PD_SUBRESULT_NO_GOODCRC | Subresult - No GoodCRC received for sent packet |
| PD_SUBRESULT_HARDRESET | Subresult - HardReset occurred. |
| PD_SUBRESULT_SOFTRESET | Subresult - SoftReset occurred. |

## Examples

```
# The following code interrupts EPR_Mode(Enter) sequence with EPR_Get_Sink_Cap message
# after receiving EPR_Mode(Enter Acknowledged)

local $SendSettings    = PD_SendPacketSettings
local $EPREnterPacket = PD_EPRModePacket
{
    Action = PD_EPR_MODE_ACTION_ENTER
}
call PD_SendPacket( $EPREnterPacket, $SendSettings )

local $GetEPRSnkCapPacket    = PD_GetEPRSinkCapMsg
local $InterruptAMSSettings = PD_InterruptAMSSettings

Call PD_InterrupAMS( $GetEPRSnkCapPacket, $InterruptAMSSettings )
```

## 4.28 PD_ReceiveCableAutoresponse

Waits to receive the specified packet type in the `$receive_Settings` parameter and, while waiting for the packet to be received, automatically responds to other packets received from the cable. This function incorporates all the features of `PD_ReceivePacket`.

### Format

```
Call PD_ReceiveCableAutoresponse( PD_ReceivePacketSettings $receive_Settings )
```

### Parameters

`$receive_Settings`

Settings for receiving packets should be of the structure type `PD_ReceivePacketSettings`, as defined in the `PD_ReceivePacket` section.

### Examples

```
$recv_settings = PD_ReceivePacketSettings
{
    WaitTimeOut = PD_DEFAULT_TIMEOUT_INFINIT
    PacketType  = PD_MESSAGE_TYPE_SOURCE_CAP
}
call PD_ReceiveCableAutoresponse($recv_settings)
```

# 5 Transaction Engine™

Power Delivery Transaction Engine™ includes high level commands and auto response capability.

## 5.1 High Level Commands

### 5.1.1 PD_SetWorkingRevision

Sets the Exerciser working revision along with Specification Revision. It should call once in whole Exerciser script. The default working revision is PD_SPEC_REVISION_2.

**Format**

```
Call PD_SetWorkingRevision( revision )
```

**Parameters**

`revision`

Indicates the target revision.

Possible values:

```
PD_SPEC_REVISION_2(default),
PD_SPEC_REVISION_3
```

**Result**

None

**Examples**

```
Call PD_SetWorkingRevision( PD_SPEC_REVISION_3 )
```

### 5.1.2 PD_SetNegotiationSetting_Source

Applies settings to power negotiation related commands as Source in PD Exerciser. If the user wants to change default settings for Source Power Negotiation, has to call this function prior calling the PD_NegotiatePower_Source or PD_NegotiatePower or PD_WaitForNegotiatePower functions to take effect.

**Format**

```
Call PD_SetNegotiationSetting_Source( PD_Negotiation_Source_Settings $settings )
```

**Parameters**

`$settings`

Defines negotiation settings for source. Should be in type of PD_Negotiation_Source_Settings template. Table below shows all available fields of PD_Negotiation_Source_Settings template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Indicates the wait timeout (micro second) to receive Request/EPRRequest Message. |
| NegotiationResponse | PD_NEGOTIATION_ACCEPT(default)<br>PD_NEGOTIATION_WAIT<br>PD_NEGOTIATION_REJECT | Indicates the response type. |
| SourceCapsRetryCount | DEFAULT_SOURCE_CAPS_RETRY_COUNT(default) | Source capabilities retry count. |
| VBusVoltage_mv | VBUS_VOLTAGE_V5_SAFE(default) | VBus voltage in millivolt. |
| SourceCapMsgSpecRev | PD_INVALID_VALUE(default)<br>Or other user defined value. | If the value is not PD_INVALID_VALUE then SourceCap Message in Negotiation sequence will be transferred using this Specification Revision. |
| AutoSpecRev | PD_FALSE,<br>PD_TRUE(default) | Rev3.0 only. Indicates whether the Exerciser should detect the Specification Revision automatically from Negotiation process or not. |

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

| | | |
|---|---|---|
| AutoUnchunkedSupport | PD_FALSE,<br>PD_TRUE(default) | Rev3.0 only. Indicates whether the Exerciser should detect the Un-chunked Support automatically from Negotiation sequence or not. |
| SkipNegotiationResponse | PD_TRUE<br>PD_FALSE(Default) | Prevents the source from sending a response back to request in the negotiation. |
| SendAcceptDelay | 0 (default) | Indicates the delay (in micro-seconds) to send ACCEPTmessage. |
| SendPSRDYDelay | 0 (default) | Indicates the delay (in micro-seconds) to send PS_RDY message. |
| SkipHardResetOnError | PD_TRUE<br>PD_FALSE(Default) | To skip sending Hard Reset during power negotiation |

**Note -** If user sets the VBusVoltage_mv, then the PD Exerciser will set VBusVoltage_mv on the VBus regardless the actual voltage value which UUT selected during the negotiation process, otherwise the Exerciser will set the VBus using the voltage which UUT selected during the negotiation process.

Note - In order to apply voltages greater than 5V, the corresponding check box should be set in recording options (*Allow VBUS > 5v*).

**Result**

None

**Examples**

```
#set negotiation using default values
$settings = PD_Negotiation_Source_Settings
call PD_SetNegotiationSetting_Source( $settings )

#set negotiation using reject as response
$settings
{
   NegotiationResponse = PD_NEGOTIATION_REJECT
}
call PD_SetNegotiationSetting_Source( $settings )
```

### 5.1.3 PD_AddSourceCap

Adds a specified Source Capability to the PD Exerciser. Before adding a group of source caps make sure that there is no unwanted source cap in the list by calling PD_ResetSourceCaps function. This function has to be called prior calling the PD_NegotiatePower_Source or PD_NegotiatePower or PD_WaitForNegotiatePower functions to take effect.

**Note** - By default there is one pre-defined source cap(vSafe5V) in the list.

**Format**

```
Call PD_AddSourceCap(PD_PowerDataObject $PowerDataObject)
```

**Parameters**

$PowerDataObject

Parameter type is PD_PowerDataObject. Refer to

> Packet PD_GetSinkCapExtendedMsg: PD_ControlMessage
>
> {
>  MessageType = PD_MESSAGE_TYPE_GET_SNK_CAP_EXT
> }

PD_SourceCapabilitiesMessage for available source power data objects.

**Result**

None

**Examples**

```
local $power_data_object = PD_PowerDataObjectFixedSupply_Source
{
   MaxCurrent_10mAUnits = 20
   Voltage_50mVUnits = 250
}
call PD_AddSourceCap($power_data_object)
```

### 5.1.4 PD_ResetSourceCaps

Clears all *Source Capabilities* defined in PD Exerciser. It should be called prior calling the `PD_AddSourceCap` function.

**Format**

```
Call PD_ResetSourceCaps()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetSourceCaps()
```

### 5.1.5 PD_NegotiatePower_Source

This command tries to establish an explicit contract as Source.

**Note**: The `PD_ResetSourceCaps`, `PD_AddSourceCap` and `PD_SetNegotiationSetting_Source` functions may need to be called before calling this function.

**Format**

```
Call PD_NegotiatePower_Source()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_REQUEST_MSG_INVALID_INDEX | Subresult - Invalid index in request message |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - Wait message has been sent as Request message response |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - Reject message has been sent as Request message response |

### Examples

```
call PD_NegotiatePower_Source()
```

## 5.1.6 PD_SetNegotiationSetting_Sink

Applies power negotiation settings as Sink. If the user wants to change default settings for Sink Power Negotiation, has to call this function prior calling the PD_NegotiatePower_Sink or PD_NegotiatePower or PD_WaitForNegotiatePower or PD_DelayAutoResponse functions to take effect.

### Format

```
Call PD_SetNegotiationSetting_Sink( PD_Negotiation_Sink_Settings $settings )
```

### Parameters

**$settings**

Should be from PD_Negotiation_Sink_Settings type.

Table below shows all available fields of PD_Negotiation_Sink_Settings template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Indicates the wait timeout(micro second) to receive SourceCapabilities Message. |
| ActiveSinkCapPos | PD_SNK_CAP_POS_AUTO_ASC (default)<br>PD_SNK_CAP_POS_AUTO_DESC<br>PD_SNK_CAP_POS_UNDEFINED<br>1..7 | Indicates which added Sink Capability by PD_AddSinkCap should be used to update current/ power and USBCommunicationCapable fields in RequestRDO.<br>When set to a number between 1 to 7, the correspond Sink Cap is used to update RequestRDO field.<br>When set to PD_SNK_CAP_POS_AUTO_ASC, the first Sink Cap matching the received Source Caps is selected automatically.<br>When set to PD_SNK_CAP_POS_AUTO_DESC, the last Sink Cap matching the received Source Caps is selected automatically.<br>When set to PD_SNK_CAP_POS_UNDEFINED, RequestRDO is left unchanged. |
| RequestRDO | 0x00000000 | Indicates the RDO which is used in the Request message. It might be updated based on the value set in ActiveSinkCapPos. |
| EPRActiveSinkCapPos | PD_SNK_CAP_POS_AUTO(default)<br>PD_SNK_CAP_POS_UNDEFINED<br>1..6 | Indicates which added Sink Capability by PD_AddEPRSinkCap should be used to update current/ power and USBCommunicationCapable fields in EPRRequestRDO.<br>When set to a number between 1 to 6, the correspond Sink Cap is used to update EPRRequestRDO field.<br>When set to PD_SNK_CAP_POS_AUTO, the first Sink Cap matching the received Source Caps is selected automatically.<br>When set to PD_SNK_CAP_POS_UNDEFINED, EPRRequestRDO is left unchanged. |
| EPRRequestRDO | 0x00000000 | Indicates the RDO which is used in the EPRRequest message. |

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

| | | |
|---|---|---|
| | | It might be updated based on the value set in EPRActiveSinkCapPos. |
| EPRRequestPDO | 0x00000000 | Indicates the PDO which is used in the EPRequest message.<br>If zero, the PDO is automatically set based on the selected Source Cap. |
| RetryCountOnWait | 2(default) | Indicates the retry count upon receiving Wait Message after sending the Request. |
| RetryDelayOnWait | 100000 us(default) | Indicates the delay time before retrying the Request, upon receiving Wait Message. |
| RequestMsgSpecRev | PD_INVALID_VALUE(default)<br>Or other user defined value. | If the value is not PD_INVALID_VALUE then Request message in Negotiation sequence will be transferred using this Specification Revision. |
| ExTriggerOnAccept | PD_FALSE(default)<br>PD_TRUE | Indicates whether to send External Trigger Out on receiving Accept message or not.<br>In M310e, it triggers connected Load Box through RS232 port. |
| ExTriggerOnPSRDY | PD_FALSE(default)<br>PD_TRUE | Indicates whether to send External Trigger Out on receiving PS_RDY message or not.<br>In M310e, it triggers connected Load Box through RS232 port. |
| AutoSpecRev | PD_FALSE,<br>PD_TRUE(default) | Rev3.0 only. Indicates whether the Exerciser should detect the Specification Revision automatically from Negotiation process or not. |
| AutoUnchunkedSupport | PD_FALSE,<br>PD_TRUE(default) | Rev3.0 only. Indicates whether the Exerciser should detect the Un-chunked Support automatically from Negotiation sequence or not. |
| SendRequestDelay | 0 (default) | Indicates the delay (in micro-seconds) to send Request message. |
| SkipHardResetOnError | PD_TRUE<br>PD_FALSE(Default) | To skip sending Hard Reset during power negotiation |
| ExTriggerDelay | 0 (default) | Indicates the delay (in micro-seconds) to send External Trigger Out. Related to ExTriggerOnAccept and ExTriggerOnPSRDY settings |

**Result**

None

**Examples**

```
#Set sink negotiation settings as default
$settings = PD_Negotiation_Sink_Settings
call PD_SetNegotiationSetting_Sink( $settings )
```

### 5.1.7 PD_AddSinkCap

Adds Sink Capabilities to PD Exerciser. Before adding a group of sink caps make sure that there is no unwanted sink cap in the list by calling PD_ResetSinkCaps function. This function has to be called prior calling the PD_NegotiatePower_Sink or PD_NegotiatePower or PD_WaitForNegotiatePower or PD_DelayAutoResponse functions to take effect.

**Note** - By default there is one pre-defined sink cap in the list.

**Format**

```
Call PD_AddSinkCap(PD_PowerDataObject $PowerDataObject)
```

**Parameters**

```
$PowerDataObject
```

Parameter type is `PD_PowerDataObject`. Refer to `PD_SPRPowerDataObjectAVS`

```
Packet PD_SPRPowerDataObjectAVS : PD_PowerDataObject
{
   MaxCurrent9To15VRange_10mAUnits      : 10 = 0
   MaxCurrent15To20VRange_10mAUnits     : 10 = 0
   SPRAVS_Rsvd_1                        : 6 = 0
   PeakCurrent                          : 2 = 0
   APDOType                             : 2 = 2
   PowerDataType                        : 2 = 3
}
```

PD_SinkCapabilitiesMessage for available sink power data objects.

**Result**

None

**Examples**

```
local $power_data_object = PD_PowerDataObjectFixedSupply_Sink
{
   OperationalCurrent_10mAUnits = 50
   Voltage_50mVUnits = 100
}
call PD_AddSinkCap($power_data_object)
```

### 5.1.8 PD_ResetSinkCaps

Clears all *Sink Capabilities* defined for PD Exercise. It should be called prior calling the `PD_AddSinkCap` function.

**Format**

```
Call PD_ResetSinkCaps()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetSinkCaps()
```

### 5.1.9 PD_NegotiatePower_Sink

Tries to establish explicit contract as Sink by sending a `Request` message.

**Note:** The `PD_ResetSinkCaps`, `PD_AddSinkCap` and `PD_SetNegotiationSetting_Sink` functions may need to be called before calling this function.

**Format**

```
Call PD_NegotiatePower_Sink()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` received as the response |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - `Wait` received as the response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `PS_RDY` message not received. |

**Examples**

```
call PD_NegotiatePower_Sink()
```

### 5.1.10 PD_WaitForNegotiatePower

Tries to establish explicit contract either as Source or Sink according to the current PD Exerciser power role. If the current power role of PD Exerciser is Source, this command will wait to receive `Request` message and if the current power role of PD Exerciser is Sink, it will wait to receive `Source_Capabilities` message.

**Note:** If the PD Exerciser is operating as Source then the **PD_ResetSourceCaps**, **PD_AddSourceCap** and **PD_SetNegotiationSetting_Source** functions may need to be called prior calling this function and if the PD Exerciser is operating as Sink then the **PD_ResetSinkCaps**, **PD_AddSinkCap** and **PD_SetNegotiationSetting_Sink** functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForNegotiatePower()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message received as the response |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - `Wait` message received as the response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Request`(PD Exerciser as Source)/`Source_Capabilities`(PD Exerciser as Sink) message not received or `PS_RDY` message not received(PD Exerciser as Sink) |

**Examples**

```
call Pd PD_WaitForNegotiatePower()
```

### 5.1.11 PD_NegotiatePower

Negotiates power with the peer port according to PD Exerciser current power role. If PD Exerciser operates as Source, this function starts power negotiation as Source and if the PD Exerciser operates as Sink, this function starts power negotiation as Sink(will wait to receive `Request` message).

**Note** - If the PD Exerciser is operating as Source then the `PD_ResetSourceCaps`, `PD_AddSourceCap` and `PD_SetNegotiationSetting_Source` functions may need to be called prior calling this function and if the PD Exerciser is operating as Sink then the `PD_ResetSinkCaps`, `PD_AddSinkCap` and `PD_SetNegotiationSetting_Sink` functions may need to be called prior calling this function.

**Format**

```
Call PD_NegotiatePower()
```

**Parameters**

None

**Result**

If PD Exerciser operates as Source this function returns same sub-results as `PD_NegotiatePower_Source` function. If PD Exerciser operates as Sink this function returns same sub-results as `PD_NegotiatePower_Sink` function.

**Examples**

```
Call PD_NegotiatePower()
```

### 5.1.12 PD_SetSwapPowerRoleSetting

It has to be called prior calling the `PD_SwapPowerRole` or `PD_WaitForSwapPowerRole` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetSwapPowerRoleSetting( Pd_SwapPowerRole_Settings $settings )
```

**Parameters**

`$settings`

Should be from `Pd_SwapPowerRole_Settings` type.

Following is the list of `Pd_SwapPowerRole_Settings` fields:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| SwapResponse | `PD_MESSAGE_TYPE_ACCEPT` (default)<br>`PD_MESSAGE_TYPE_WAIT`<br>`PD_MESSAGE_TYPE_REJECT`<br>`PD_RESPONSE_NOT_SUPPORTED` (PD3.0 only) | Defines the response type. |
| SkipSwap | `PD_TRUE`<br>`PD_FALSE`(Default) | If set to `PD_TRUE`, `PD_SwapPowerRole` AMS will not swap the power role. |
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Timeout(in micro-seconds) to wait in order to receive the `PR_SWAP` message. |
| RetryCountOnWait | 2(default) | Indicates the retry count after receiving `Wait` message in response to sent `PR_Swap` Message. |
| RetryDelayOnWait | 100000(default) | Indicates the retry delay time(in micro-seconds) upon receiving `Wait` message in response to sent `PR_Swap` Message. |

| | | |
|---|---|---|
| SwapResponseOnWait | PD_MESSAGE_TYPE_ACCEPT (default)<br>PD_MESSAGE_TYPE_WAIT<br>PD_MESSAGE_TYPE_REJECT<br>PD_RESPONSE_NOT_SUPPORTED (PD3.0 only) | Defines the response type in a case that the DUT initially responded to PR_Swap message with a Wait message and after a while issued a PR_Swap message towards the PD Exerciser. |
| SkipSendingPSRDY | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, PD_SwapPowerRole AMS will not send the PS_RDY message. |
| NewSrcRpType | CC_RP_CUR_DEFAULT<br>CC_RP_CUR_1_5 (Default)<br>CC_RP_CUR_3_0 | Indicates the Rp current value when swapping from Sink to Source. |
| SendPSRDYDelay | 0 (default) | Indicates the delay (in micro-seconds) to send PS_RDY message. |
| SkipWaitingForVBus | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, will skip waiting for VBus On/Off operations. |
| NoGoodCRCToPSRDY | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, PD_SwapPowerRole AMS will not send the GoodCRC for PS_RDY message. |
| SkipPowerNegotiation | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, will skip power negotiation after the power swap. |

**Result**

None

**Examples**

```
#Set response to REJECT;
#Set response on receiving Wait message to ACCEPT;
#Set Rp type while swapping to source as CC_RP_CUR_3_0;
$settings = Pd_SwapPowerRole_Settings
{
   SwapResponse = PD_MESSAGE_TYPE_REJECT
   SwapResponseOnWait = PD_MESSAGE_TYPE_ACCEPT
   NewSrcRpType = CC_RP_CUR_3_0
}
call PD_SetSwapPowerRoleSetting( $settings )
```

### 5.1.13 PD_SwapPowerRole

Tries to swap power role. It will start Swap Power Role AMS.

**Note:** The PD_SetSwapPowerRoleSetting function may need to be called prior calling this function.

**Format**

```
Call PD_SwapPowerRole()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - Response not received |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - Reject message received as response |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - Wait message received as response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - PS_RDY message not received(PD Exerciser as Sink) |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message received as response |

**Examples**

```
call PD_SwapPowerRole()
```

### 5.1.14 PD_WaitForSwapPowerRole

Waits to receive PR_Swap message and will respond to incoming messages as part of the *Swap Power Role* AMS.

**Note:** The PD_SetSwapPowerRoleSetting function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForSwapPowerRole()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - Wait message has been sent as response |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - Reject message has been sent as response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - PR_Swap message not received or PS_RDY message not received(PD Exerciser as Sink) |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message has been sent as response |

**Examples**

```
call PD_WaitForSwapPowerRole()
```

### 5.1.15 Pd_SetFRSwapNewSnkSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the PD_FastRoleSwap function to take effect.

**Format**

```
Call Pd_SetFRSwapNewSnkSetting( Pd_FRSwapNewSnkSettings $fr_swap_settins )
```

**Parameters**

$fr_swap_settins

Should be from Pd_FRSwapNewSnkSettings type. Following are the available fields for this packet template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| NoGoodCRCToPSRDY | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to send a GoodCRC message to receiving PS_RDY message or not. |
| SkipSendingPSRDY | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to skip sending PS_RDY message or not. |
| NoGoodCRCToFRSwap | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to send a GoodCRC message to receiving FR_Swap message or not. |

| | | |
|---|---|---|
| SkipSendingAccept | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to skip sending Accept message or not. |

**Result**

None

**Examples**

```
$settings = Pd_FRSwapNewSnkSettings
{
   NoGoodCRCToFRSwap = PD_TRUE
}
call Pd_SetFRSwapNewSnkSetting( $settings )
```

### 5.1.16 Pd_SetFRSwapNewSrcSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the PD_WaitForFRSwapSignal or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call Pd_SetFRSwapNewSrcSetting( Pd_FRSwapNewSrcSettings $fr_swap_settins )
```

**Parameters**

$fr_swap_settins

Should be from Pd_FRSwapNewSrcSettings type. Following are the available fields for this packet template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| NoGoodCRCToPSRDY | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to send a GoodCRC message to receiving PS_RDY message or not. |
| SkipSendingPSRDY | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to skip sending PS_RDY message or not. |
| VBusVoltageThreshold_mV | VBUS_VOLTAGE_V5_SAFE_MAX(Default) | If the VBus drops below this value, the new Source(PD Exerciser) will turn on the VBus. |
| NewVBusVoltage_mV | VBUS_VOLTAGE_V5_SAFE(Default) | PD Exerciser as new Source will increase the VBus voltage up-to this value. |
| FRSwapSignalWaitTimeout | 1000000(Default) | Timeout to wait for receiving the FR_Swap Signal (micro seconds). |
| FRSwapNewSrcRpCurrent | CC_RP_CUR_1_5(Default) | PD Exerciser as new Source will apply this RpCurrent value. |
| NoGoodCrcToAccept | PD_TRUE<br>PD_FALSE(Default) | Indicates whether to send a GoodCRC message to receiving Accept message or not. |

**Result**

None

**Examples**

```
$settings = Pd_FRSwapNewSrcSettings
{
   VBusVoltageThreshold_mV = VBUS_VOLTAGE_V5_SAFE
   NoGoodCrcToAccept = PD_TRUE
}
call Pd_SetFRSwapNewSrcSetting( $settings )
```

### 5.1.17 PD_FastRoleSwap

Applicable to PD Rev 3.0 only. Sends the *FastRoleSwap* Signal and handles *Fast Role Swap* AMS.

**Note -** The Pd_SetFRSwapNewSnkSetting function may need to be called prior calling this function.

**Format**

```
Call PD_FastRoleSwap()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded. |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |

**Examples**

```
Call PD_FastRoleSwap()
```

### 5.1.18 PD_WaitForFRSwapSignal

Applicable to PD Rev 3.0 only. Waits for a specified time to receive the *FR_Swap Signal*. During this time, any received messages will be handeled.

**Note 1-** Received *FastRoleSwap* Signal will handle by *FastRoleSwap Event Handler* automatically.

**Note 2-** The `Pd_SetFRSwapNewSrcSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForFRSwapSignal()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_WaitForFRSwapSignal()
```

### 5.1.19 Pd_GetPPSStatus

Applicable to PD Rev 3.0 only. Sends `Get_PPS_Status` message and starts the *GetPPSStatus* AMS.

**Format**

```
Call Pd_GetPPSStatus()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - Response not received |
| PD_SUBRESULT_UNEXPECTED_MSG_RECEIVED | Subresult - Unexpected packet received |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported has been received as response |

**Examples**

```
Call Pd_GetPPSStatus()
```

### 5.1.20 Pd_SetGetPPSStatusSetting

Applicable to PD Rev 3.0 only. It must be called prior to call Pd_WaitForGetPPSStatus or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetGetPPSStatusSetting( PD_GetPPSStatus_Settings $settings )
```

**Parameters**

$settings

Parameter type is PD_GetPPSStatus_Settings. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(micro second) to receive Get_PPS_Status message. |
| ResponseType | PD_RESPONSE_NOT_SUPPORTED, PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the Get_PPS_Status message. |

**Result**

None

**Examples**

```
$get_ppsstatus_setting = PD_GetPPSStatus_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetPPSStatusSetting( $get_ppsstatus_setting )
```

### 5.1.21 Pd_SetPPSStatusDataBlock

Applicable to PD Rev 3.0 only. Sets the *PPS Status Data Block* in PD Exerciser. It must be called before Pd_WaitForGetPPSStatus or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetPPSStatusDataBlock( PD_PPSStatusDataBlock $pps_status_db )
```

**Parameters**

$pps_status_db

Parameter type is PD_PPSStatusDataBlock. Refer to

PD_PPSStatusMsg for available fields.

**Result**

None

**Examples**

```
$pps_status_db = PD_PPSStatusDataBlock
Call PD_SetPPSStatusDataBlock( $pps_status_db )
```

### 5.1.22 Pd_ResetPPSStatusDataBlock

Applicable to PD Rev 3.0 only. Clears the *PPS Status Data Block* in PD Exerciser. Should be called prior calling the `Pd_SetPPSStatusDataBlock` function.

**Format**

```
Call PD_ResetPPSStatusDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResetPPSStatusDataBlock()
```

### 5.1.23 Pd_WaitForGetPPSStatus

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Get_PPS_Status` message. It will respond to incoming messages as part of *GetPPSStatus* AMS.

**Note:** The `Pd_ResetPPSStatusDataBlock`, `Pd_SetPPSStatusDataBlock` and `Pd_SetGetPPSStatusSetting` functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGetPPSStatus( )
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_PPS_Status` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call PD_WaitForGetPPSStatus()
```

### 5.1.24 Pd_SetPPSNegotiationSetting_Sink

Applicable to PD Rev 3.0 only. Applies power negotiation settings as a Sink which is communicating with a PPS Source. If the user wants to change default settings for Sink Power Negotiation, has to call this function prior to call `Pd_StartPPSNegotiatePower_Sink` and `Pd_NextPPSNegotiatePower_Sink` functions to take effect. This function is not supported by `PD_DelayAutoResponse` function.

#### Format

```
Call PD_SetPPSNegotiationSetting_Sink( PD_PPSNegotiation_Sink_Settings $settings )
```

#### Parameters

`$settings`

Should be from `PD_PPSNegotiation_Sink_Settings` type. `Pd_PPSNegotiation_Sink_Settings` template contains all fields of `Pd_Negotiation_Sink_Settings`(Refer to `PD_SetNegotiationSetting_Sink`) and some extra fields which are listed in table below:

| Field Name | Default Values | Description |
|---|---|---|
| MinOutputVolt_20mVUnits | 0x00 | Min output voltage in 20mV units. Output Voltage will be increased from this value. Refer to `PD_ProgrammableRDO`. |
| MaxOutputVolt_20mVUnits | 0x00 | Max output voltage in 20mV units. Output Voltage will be decreased from this value. Refer to `PD_ProgrammableRDO`. |
| OperatingCur_50mAUnits | 0x00 | Refer to `PD_ProgrammableRDO`. |
| UnchunkedExtMsgSupported | 1 | Refer to `PD_ProgrammableRDO`. |
| NoUsbSuspend | 0 | Refer to `PD_ProgrammableRDO`. |
| UsbCommunicationsCapable | 0 | Refer to `PD_ProgrammableRDO`. |
| CapabilityMismatch | 0 | Refer to `PD_ProgrammableRDO`. |
| ObjectPosition | 0 | Refer to `PD_ProgrammableRDO`. |
| VoltageStep_20mVUnits | 0x01 | Voltage Step in 20mV units. |
| IncrementalStep | `PD_TRUE` (increasing) | Indicates whether the Voltage is increasing or decreasing. |

#### Result

None

#### Examples

```
#Set pps negotiation sink settings
$pps_nego_settings = Pd_PPSNegotiation_Sink_Settings
{
    MinOutputVolt_20mVUnits = 500
    MaxOutputVolt_20mVUnits = 500
    OperatingCur_50mAUnits = 10
    ObjectPosition = 1
    VoltageStep_20mVUnits = 2
    IncrementalStep = PD_FALSE
}
call Pd_SetPPSNegotiationSetting_Sink($pps_nego_settings)
```

### 5.1.25 Pd_StartPPSNegotiatePower_Sink

Applicable to PD Rev 3.0 only. Starts *PPS Power Negotiation* as a Sink. This function should be called before `Pd_NextPPSNegotiatePower_Sink`.

**Note:** The `Pd_SetPPSNegotiationSetting_Sink` function may need to be called prior calling this function.

#### Format

```
Call PD_StartPPSNegotiatePower_Sink()
```

**Parameters**

None

**Result**

Refer to PD_NegotiatePower_Sink.

**Examples**

```
call Pd_StartPPSNegotiatePower_Sink()

PD_Loop(10)
{
    call Pd_NextPPSNegotiatePower_Sink()
    call PD_DelayAutoResponse(100000)
}
```

### 5.1.26 Pd_NextPPSNegotiatePower_Sink

Applicable to PD Rev 3.0 only. Next *PPS Power Negotiation* as a Sink with incremented or decremented output voltage value. This function should be called after Pd_StartPPSNegotiatePower_Sink.

**Note:** The Pd_SetPPSNegotiationSetting_Sink function may need to be called prior calling this function.

**Format**

```
Call PD_NextPPSNegotiatePower_Sink()
```

**Parameters**

None

**Result**

Refer to PD_NegotiatePower_Sink.

**Examples**

```
call Pd_StartPPSNegotiatePower_Sink()

PD_Loop(10)
{
    call Pd_NextPPSNegotiatePower_Sink()
    call PD_DelayAutoResponse(100000)
}
```

### 5.1.27 PD_SetDataResetSetting

Applicable to PD Rev 3.0 only. It must be called prior to call PD_WaitForDataReset or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call Pd_SetDataResetSetting( Pd_DataReset_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is PD_DataReset_Settings. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| DataResetWaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(micro second) to receive Pd_DataResetMessage. |

| | | |
|---|---|---|
| DataResetResponse | PD_RESPONSE_REJECT,<br>PD_RESPONSE_ACCEPT(default) | Indicates response upon receiving the Pd_DataResetMessage. |
| SkipSendingPSRDY | PD_TRUE,<br>PD_FALSE (default) | To skip sending PSRdy packet. |
| RunningUSBExercisers | PD_USB2_EXERCISER (default)<br>PD_USB3_EXERCISER,<br>PD_USB4_EXERCISER,<br>PD_TBT3_EXERCISER, | Exercisers to be run in Data Reset process |
| SkipErrorRecovery | PD_TRUE,<br>PD_FALSE (default) | To skip Error Recovery |
| SkipComplete | PD_TRUE,<br>PD_FALSE (default) | To skip Completion |

**Result**

None

**Examples**

```
$send_data_reset = PD_DataReset_Settings
{
   DataResetResponseType = PD_RESPONSE_REJECT
    RunningUSBExercisers = PD_USB2_EXERCISER | PD_USB3_EXERCISER
}
Call PD_SetDataResetSetting( $send_data_reset )
```

### 5.1.28 PD_SendDataReset

Applicable to PD Rev 3.0 only. Sends `Pd_DataResetMessage` and starts the *DataReset* AMS.

**Format**

```
Call PD_SendDataReset()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - Response not received |
| PD_SUBRESULT_UNEXPECTED_MSG_RECEIVED | Subresult - `Unexpected` message received as response |

**Examples**

```
call PD_SendDataReset()
```

### 5.1.29 PD_WaitForDataReset

Waits for user-defined timeout to receive `Pd_DataResetMessage` and will respond to incoming messages as part of the *DataReset* AMS.

**Note:** The `PD_SetDataResetSetting` function may need to be called prior calling this function.

### Format

```
Call PD_WaitForDataReset()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `DataReset` message not received |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response |

### Examples

```
call PD_WaitForDataReset()
```

## 5.1.30 PD_SetSwapDataRoleSetting

It must be called before **PD_SwapDataRole** or **PD_WaitForSwapDataRole** or **PD_DelayAutoResponse** functions to take effect.

### Format

```
Call PD_SetSwapDataRoleSetting( Pd_SwapDataRole_Settings $settings )
```

### Parameters

`$settings`

Should be from `Pd_SwapDataRole_Settings` type. Table below shows all fields of the `Pd_SwapDataRole_Settings` template which are applied to *DR_Swap AMS*:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| SwapResponse | `PD_MESSAGE_TYPE_ACCEPT`(default)<br>`PD_MESSAGE_TYPE_REJECT`<br>`PD_MESSAGE_TYPE_WAIT`<br>`PD_RESPONSE_NOT_SUPPORTED` (PD3.0 only) | Defines the response type. |
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Timeout(in micro-seconds) to wait in order to receive `DR_SWAP` message. |
| RetryCountOnWait | 2 (default) | Indicates the retry count after receiving Wait Message in response to sent DR_Swap Message. |
| RetryDelayOnWait | 100000 (default) | Indicates the retry delay time(in micro-seconds) upon receiving Wait Message in response to sent DR_Swap Message. |
| SwapResponseOnWait | `PD_MESSAGE_TYPE_ACCEPT` (default)<br>`PD_MESSAGE_TYPE_WAIT`<br>`PD_MESSAGE_TYPE_REJECT`<br>`PD_RESPONSE_NOT_SUPPORTED` (PD3.0 only) | Defines the response type in a case that the DUT initially responded to DR_Swap message with a Wait message and after a while issued a DR_Swap message towards the PD Exerciser. |

### Result

None

**Examples**

```
#Set response to REJECT;
#Set response on receiving Wait message to ACCEPT;
$settings = Pd_SwapDataRole_Settings
{
   SwapResponse = PD_MESSAGE_TYPE_REJECT
   SwapResponseOnWait = PD_MESSAGE_TYPE_ACCEPT
}
call PD_SetSwapDataRoleSetting( $settings )
```

### 5.1.31 PD_SwapDataRole

Tries to swap the data role. It will start the *Swap Data Role AMS*.

**Note:** The `PD_SetSwapDataRoleSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_SwapDataRole()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - Response not received |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been received |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - `Wait` message has been received |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message received as response |

**Examples**

```
call PD_SwapDataRole()
```

### 5.1.32 PD_WaitForSwapDataRole

Waits for user-defined timeout to receive `DR_Swap` message and will respond to incoming messages as part of the *Swap Data Role* AMS.

**Note:** The `PD_SetSwapDataRoleSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForSwapDataRole()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - Reject message has been sent as response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DR_Swap message not received |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message has been sent as response |

### Examples

```
call PD_WaitForSwapDataRole()
```

## 5.1.33 PD_SetSwapVConnSetting

It has to be called prior calling the PD_SwapVConn or PD_WaitForSwapVConn or PD_DelayAutoResponse functions to take effect.

### Format

```
Call PD_SetSwapVConnSetting( Pd_SwapVConn_Settings $settings )
```

### Parameters

$settings

It should be from Pd_SwapVConn_Settings type. The table below shows the Pd_SwapVConn_Settings template fields:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| SwapResponse | PD_MESSAGE_TYPE_ACCEPT(default)<br>PD_MESSAGE_TYPE_REJECT<br>PD_MESSAGE_TYPE_WAIT<br>PD_RESPONSE_NOT_SUPPORTED (PD3.0 only) | Defines the response type. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Timeout(in micro-seconds) to wait in order to receive VCONN_SWAP message. |
| SkipSwap | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, the command skips VConn swap. |
| SkipSendingPSRDY | PD_TRUE<br>PD_FALSE(Default) | If set to PD_TRUE, PD_SwapVConn AMS will not send the PS_RDY message. |
| RetryCountOnWait | 2 (default) | Indicates the retry count after receiving Wait Message in response to sent VConn_Swap Message. |
| RetryDelayOnWait | 100000 (default) | Indicates the retry delay time(in micro-seconds) upon receiving Wait Message in response to sent VConn_Swap Message. |
| SwapResponseOnWait | PD_MESSAGE_TYPE_ACCEPT (default)<br>PD_MESSAGE_TYPE_WAIT<br>PD_MESSAGE_TYPE_REJECT<br>PD_RESPONSE_NOT_SUPPORTED (PD3.0 only) | Defines the response type in a case that the DUT initially responded to VConn_Swap message with a Wait message and after a while issued a VConn_Swap message towards the PD Exerciser. |
| VConnVoltage_mV | PD_INVALID_VALUE (default)<br>0 – 5.5 V | If set to PD_INVALID_VALUE, 5V is sourced for Vconn. |

### Result

None

### Examples

```
#Set the response to REJECT;
#skip swapping in the case that PD Exerciser is initiator;
$settings = Pd_SwapVConn_Settings
{
   SwapResponse = PD_MESSAGE_TYPE_REJECT
```

```
    SkipSwap = PD_TRUE
}
call PD_SetSwapVConnSetting( $settings )
```

### 5.1.34 PD_SwapVConn

Tries to swap VConn. It will start the *Swap VConn* AMS.

**Note:** The `PD_SetSwapVConnSetting` function may need to be called prior calling this function.

#### Format

```
Call PD_SwapVConn()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - Response not received |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been received |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - `Wait` message has been received |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `PS_RDY` message not received(PD Exerciser as VCONN Source) |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message message has been received |

#### Examples

```
call PD_SwapVConn()
```

### 5.1.35 PD_WaitForSwapVConn

Waits for user-defined timeout to receive `VCONN_Swap` message and will respond to incoming messages as part of *Swap VConn* AMS.

**Note:** The `PD_SetSwapVConnSetting` function may need to be called prior calling this function.

#### Format

```
Call PD_WaitForSwapVConn()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |

| | |
|---|---|
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - VCONN_Swap message not received |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - Wait message has been sent as response |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - Reject message has been sent as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message has been sent as response |

**Examples**

```
call PD_WaitForSwapVConn()
```

### 5.1.36 PD_SetGotoMinSetting

It has to be called prior to call PD_WaitForGotoMin or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetGotoMinSetting( PD_GotoMin_Settings $settings )
```

**Parameters**

`$settings`

Setting type is PD_GotoMin_Settings. Available fields of this type are:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait timeout(micro second) for receiving GotoMin message. |
| ResponseType | PD_RESPONSE_UNSPECIFIED(default), PD_RESPONSE_NOT_SUPPORTED | Indicates the response type upon receiving GotoMin message. |

**Result**

None

**Examples**

```
$gotomin_setting = PD_GotoMin_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGotoMinSetting( $gotomin_setting )
```

### 5.1.37 PD_GotoMin

Starts the *GotoMin* AMS.

**Format**

```
Call PD_GotoMin()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |

| | |
|---|---|
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket are valid also (depends on the error type which has been occurred during sending data). |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message has been received |

**Examples**

```
call PD_GotoMin()
```

### 5.1.38 PD_WaitForGotoMin

Waits for user-defined timeout to receive GotoMin message and will respond to incoming messages as part of *GotoMin* AMS.

**Note:** The PD_SetGotoMinSetting function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGotoMin()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - GotoMin or PS_RDY message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - Not_Supported message has been sent as response |

**Examples**

```
call PD_WaitForGotoMin()
```

### 5.1.39 PD_SetGetSourceCapSetting

It has to be called prior to call PD_WaitForGetSourceCapabilities or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetGetSourceCapSetting( PD_GetSourceCapability_Settings $settings )
```

**Parameters**

`$settings`

Setting type is PD_GetSourceCapability_Settings. Available fields of this type are:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait timeout(micro second) for receiving GetSourceCap message. |
| ResponseType | PD_RESPONSE_UNSPECIFIED(default), PD_RESPONSE_NOT_SUPPORTED | Indicates the response type upon receiving GetSourceCap message. |

| | | |
|---|---|---|
| SkipNegotiationAsSink | PD_FALSE(default),<br>PD_TRUE | Enables the sink to skips negotiation after receiving SourceCap message. |

**Result**

None

**Examples**

```
$getsrccap_setting = PD_GetSourceCapability_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetSourceCapSetting( $getsrccap_setting )
```

### 5.1.40 PD_GetSourceCapabilities

Starts GetSourceCapabilities AMS.

**Format**

```
Call PD_GetSourceCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No messages received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message received as response |

**Examples**

```
Call PD_GetSourceCapabilities()
```

### 5.1.41 PD_WaitForGetSourceCapabilities

Waits for user-defined timeout to receive `Get_Source_Cap` message. It will respond to incoming messages as part of the *Get_Source_Cap* AMS.

**Note:** The `PD_SetGetSourceCapSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGetSourceCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Source_Cap` message not received |
| PD_SUBRESULT_REQUEST_MSG_INVALID_INDEX | Subresult - Invalid index in request message |
| PD_SUBRESULT_RESPONSE_WAIT | Subresult - `Wait` message has been sent as request message response |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as request message response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message has been sent as response |

**Examples**

```
Call PD_WaitForGetSourceCapabilities()
```

### 5.1.42 PD_SetGetSinkCapSetting

It has to be called prior to call `PD_WaitForGetSinkCapabilities` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetGetSinkCapSetting( PD_GetSinkCapability_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `PD_GetSinkCapability_Settings`. For available fields of this type are:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait timeout (micro second) for receiving GetSourceCap message. |
| ResponseType | `PD_RESPONSE_UNSPECIFIED`(default), `PD_RESPONSE_NOT_SUPPORTED` | Indicates the response type upon receiving GetSourceCap message. |
| NoGoodCRCToSnkCap | `PD_FALSE`(default), `PD_TRUE` | Prevents the source device from sending GoodCRC for the SinkCap messages. |

**Result**

None

**Examples**

```
$getsnkcap_setting = PD_GetSinkCapability_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetSinkCapSetting( $getsnkcap_setting )
```

### 5.1.43 PD_GetSinkCapabilities

Starts the GetSinkCapabilities AMS.

**Format**

```
Call PD_GetSinkCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message received as response |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No messages received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message received as response |

**Examples**

```
Call PD_GetSinkCapabilities()
```

### 5.1.44 PD_WaitForGetSinkCapabilities

Waits for user-defined timeout to receive `Get_Sink_Cap` message. It will respond to incoming messages as part of *GetSinkCapabilities* AMS.

**Note:** The `PD_SetGetSinkCapSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGetSinkCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Sink_Cap` message not received. |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Rev3.0 only. Subresult - `Not_Supported` message has been sent as response |

**Examples**

```
Call PD_WaitForGetSinkCapabilities()
```

### 5.1.45 PD_SendBISTCarrierMode

Starts BISTCarrierMode AMS.

**Format**

```
Call PD_SendBISTCarrierMode(OrderedSetType)
```

### Parameters

`OrderedSetType`

Indicates the Ordered Set type

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed |

### Examples

```
Call PD_SendBISTCarrierMode(PD_ORDERED_SET_TYPE_SOP)
```

## 5.1.46 PD_SendBISTTestData

Starts *BISTTestData* AMS.

### Format

```
Call PD_SendBISTTestData( OrderedSetType, PD_BISTTestData $test_data )
```

### Parameters

`OrderedSetType`

Indicates the Ordered Set type

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`$test_data`

Defines the Test Data to be sent to the UUT

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` are valid also (depends on the error type which has been occurred during sending data). |

### Examples

```
$test_data = PD_BISTTestData
{
    TestData = { 00 00 00 00
                 AA AA AA AA
                 AA AA 00 00
                 AA AA AA AA
                 00 00 AA AA
                 AA AA AA AA }
}

Call PD_SendBISTTestData( PD_ORDERED_SET_TYPE_SOP_PRIME, $test_data )
```

### 5.1.47 PD_GetSourceCapExtended

Applicable to PD Rev 3.0 only. Starts *GetSourceCapExtended* AMS.

#### Format

```
Call PD_GetSourceCapExtended()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

#### Examples

```
Call PD_GetSourceCapExtended()
```

### 5.1.48 PD_SetGetSrcCapExtSetting

Applicable to PD Rev 3.0 only. It has to be called prior to call `PD_WaitForGetSrcCapExtended` or `PD_DelayAutoResponse` functions.

#### Format

```
Call PD_SetGetSrcCapExtSetting( PD_GetSourceCapExtented_Settings $settings )
```

#### Parameters

`$settings`

Setting type is `PD_GetSourceCapExtented_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(micro second) to receive `Get_Source_Cap_Extended` message. |

| | | Default: |
|---|---|---|
| ResponseType | PD_RESPONSE_NOT_SUPPORTED, PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the Get_Source_Cap_Extended message. |
| SkipSrcCapExt | PD_TRUE, PD_FALSE(default) | Indicates whether to skip sending Source_Capabilities_Extended message or not. |
| SendSrcCapExtDelay | 0(default) | Defines the delay before sending Source_Capabilities_Extended message. |

**Result**

None

**Examples**

```
$getsrccapext_setting = PD_GetSourceCapExtented_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetSrcCapExtSetting( $getsrccapext_setting )
```

### 5.1.49 PD_WaitForGetSrcCapExtended

Applicable to PD Rev 3.0 only. Wait for user-defined timeout to receive *Get_Source_Cap_Extended* message. It will respond to incoming messages as part of *GetSourceCapExtended* AMS.

**Note:** The PD_SetGetSrcCapExtSetting, PD_SetSrcCapExtDataBlock and PD_ResetSrcCapExtDataBlock functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGetSrcCapExtended()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - Get_Source_Cap_Extended message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message sent as response. |

**Examples**

```
Call PD_WaitForGetSrcCapExtended()
```

### 5.1.50 PD_SetSrcCapExtDataBlock

Applicable to PD Rev 3.0 only. Sets *Source Capabilities Extended Data Block* in PD Exerciser. It has to be called prior calling the PD_WaitForGetSrcCapExtended or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetSrcCapExtDataBlock( PD_SourceCapExtDataBlock $src_cap_ext )
```

**Parameters**

`$src_cap_ext`

parameter type is `PD_SourceCapExtDataBlock`. Refer to `PD`_SourceCapExtendedMsg for available data fields.

**Result**

None

**Examples**

```
$src_cap_ext = PD_SourceCapExtDataBlock
Call PD_SetSrcCapExtDataBlock( $src_cap_ext )
```

### 5.1.51 PD_ResetSrcCapExtDataBlock

Applicable to PD Rev 3.0 only. Clears the *Source Capabilities Extended Data Block* in PD Exerciser. It should be called before calling the `PD_SetSrcCapExtDataBlock` function.

**Format**

```
Call PD_ResetSrcCapExtDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResetSrcCapExtDataBlock()
```

### 5.1.52 PD_GetStatus

Applicable to PD Rev 3.0 only. Starts the *GetStatus* AMS.

**Format**

```
Call PD_GetStatus()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

### Examples

```
Call PD_GetStatus()
```

## 5.1.53 PD_SetGetStatusSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForGetStatus` or `PD_DelayAutoResponse` functions to take effect.

### Format

```
Call PD_SetGetStatusSetting( PD_GetStatus_Settings $settings )
```

### Parameters

`$settings`

Parameter type is `PD_GetStatus_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(micro second) to receive `Get_Status` message. |
| ResponseType | PD_RESPONSE_NOT_SUPPORTED, PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the `Get_Status` message. |

### Result

None

### Examples

```
$getstatus_setting = PD_GetStatus_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetStatusSetting( $getstatus_setting )
```

## 5.1.54 PD_WaitForGetStatus

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Get_Status` message. It will respond to incoming messages as part of *GetStatus* AMS.

**Note:** The `PD_SetGetStatusSetting`, `PD_SetStatusDataBlock` and `PD_ResetStatusDataBlock` functions may need to be called prior calling this function.

### Format

```
Call PD_WaitForGetStatus( )
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |

| | |
|---|---|
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - Get_Status message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message sent as response. |

**Examples**

```
Call PD_WaitForGetStatus()
```

### 5.1.55 PD_SetStatusDataBlock

Applicable to PD Rev 3.0 only. Sets the *Status Data Block* in PD Exerciser. It has to be called prior calling the PD_WaitForGetStatus or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetStatusDataBlock( PD_StatusDataBlock $status_db )
```

**Parameters**

$status_db

Parameter type is PD_StatusDataBlock. Refer to

PD_StatusMsg for available fields.

**Result**

None

**Examples**

```
$status_db = PD_StatusDataBlock
Call PD_SetStatusDataBlock( $status_db )
```

### 5.1.56 PD_ResetStatusDataBlock

Applicable to PD Rev 3.0 only. Clears the *Status Data Block* in PD Exerciser. It should be called prior calling the PD_SetStatusDataBlock function.

**Format**

```
Call PD_ResetStatusDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResetStatusDataBlock()
```

### 5.1.57 PD_GetBatteryStatus

Applicable to PD Rev 3.0 only. Starts the *GetBatteryStatus* AMS.

**Note:** The PD_SetGetBatteryStatusDataBlock may need to be called prior calling this function.

**Format**

```
Call PD_GetBatteryStatus()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call PD_GetBatteryStatus()
```

### 5.1.58 PD_SetGetBatteryStatusDataBlock

Applicable to PD Rev 3.0 only. Sets the *GetBatteryStatus Data Block* in PD Exerciser. It has to be called before `PD_GetBatteryStatus` function to take effect.

**Format**

```
Call PD_SetGetBatteryStatusDataBlock( PD_GetBatteryStatusDataBlock $get_battery_stat_db )
```

**Parameters**

`$get_battery_stat_db`

Parameter type is `PD_GetBatteryStatusDataBlock`. Refer to `PD_GetBatteryStatusMsg` for available fields.

**Result**

None

**Examples**

```
$get_battery_stat_db = PD_GetBatteryStatusDataBlock
Call PD_SetGetBatteryStatusDataBlock( $get_battery_stat_db )
```

### 5.1.59 PD_SetGetBatteryStatusSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForGetBatteryStatus` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetGetBatteryStatusSetting( PD_GetBatteryStatus_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is `PD_GetBatteryStatus_Settings`. Available fields of this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(micro second) to receive `Get_Battery_Status` message. |
| ResponseType | `PD_RESPONSE_NOT_SUPPORTED`, `PD_RESPONSE_UNSPECIFIED`(default) | Indicates response upon receiving the `Get_Battery_Status` message. |

**Result**

None

**Examples**

```
$getbattstatus_setting = PD_GetBatteryStatus_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetBatteryStatusSetting( $getbattstatus_setting )
```

### 5.1.60 PD_WaitForGetBatteryStatus

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Get_Battery_Status` message. It will respond to incoming messages as part of *GetBatteryStatus* AMS.

**Note:** The `PD_SetGetBatteryStatusSetting`, `PD_SetBatteryStatusDO` and `PD_ResetBatteryStatusDO` functions may need to be called prior to call this function.

**Format**

```
Call PD_WaitForGetBatteryStatus()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Battery_Status` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call PD_WaitForGetBatteryStatus()
```

### 5.1.61 PD_SetBatteryStatusDO

Applicable to PD Rev 3.0 only. Sets the *BatteryStatus Data Object* in PD Exerciser. It has to be called prior calling the `PD_WaitForGetBatteryStatus` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetBatteryStatusDO( PD_BatteryStatusDataObject $battery_status )
```

**Parameters**

`$battery_status`

Parameter type is `PD_BatteryStatusDataObject`. Refer to `PD_BatteryStatusMsg` for available fields of this type.

**Result**

None

**Examples**

```
$battery_status = PD_BatteryStatusDataObject
Call PD_SetBatteryStatusDO( $battery_status )
```

### 5.1.62 PD_ResetBatteryStatusDO

Applicable to PD Rev 3.0 only. Clears the *BatteryStatus Data Object* in PD Exerciser. It should be called prior to call the `PD_SetBatteryStatusDO` function.

**Format**

```
Call PD_ResetBatteryStatusDO()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResetBatteryStatusDO()
```

### 5.1.63 PD_Alert

Applicable to PD Rev 3.0 only. Starts *Alert* AMS.

**Note:** The `PD_SetAlertDO` function may need to be called prior to call this function.

**Format**

```
Call PD_Alert()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` are valid also (depends on the error type which has been occurred during sending data). |

### Examples

```
Call PD_Alert()
```

## 5.1.64 PD_SetAlertDO

Applicable to PD Rev 3.0 only. Sets *Alert Data Object* in PD Exerciser. It has to be called prior calling the `PD_Alert` function to take effect.

### Format

```
Call PD_SetAlertDO( PD_AlertDataObject $alert_do )
```

### Parameters

`$alert_do`

Parameter type is `PD_AlertDataObject`. Refer to `PD_AlertMsg` for available fields of this type.

### Result

None

### Examples

```
$alert_do = PD_AlertDataObject
Call PD_SetAlertDO( $alert_do )
```

## 5.1.65 PD_SetAlertSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForAlert` or `PD_DelayAutoResponse` functions to take effect.

### Format

```
Call PD_SetAlertSetting( PD_Alert_Settings $settings )
```

### Parameters

`$settings`

Parameter type is `PD_Alert_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(micro second) to receive `Alert` message. |
| ResponseType | `PD_RESPONSE_NO_RESPONSE`, `PD_RESPONSE_NOT_SUPPORTED`, `PD_RESPONSE_UNSPECIFIED`(default) | If it is set to `PD_RESPONSE_UNSPECIFIED`, the response will be either a `GetStatus` or `GetBatteryStatus` based on the ADO in the received Alert message. |
| DelayBeforeResponse | `0` (default) | Indicates the delay (in micro-seconds) to send `Response` message. |

### Result

None

### Examples

```
$alert_setting = PD_Alert_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetAlertSetting( $alert_setting )
```

### 5.1.66 PD_WaitForAlert

Applicable to PD Rev 3.0 only. Waits for a user-defined timeout to receive `Alert` message. It will respond to incoming messages as part of the *Alert* AMS.

**Note:** The `PD_SetAlertSetting` function may need to be called prior to call this function.

#### Format

```
Call PD_WaitForAlert()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Alert` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

#### Examples

```
Call PD_WaitForAlert()
```

### 5.1.67 PD_GetBatteryCap

Applicable to PD Rev 3.0 only. Starts the *GetBatteryCap* AMS.

**Note:** The `PD_SetGetBatteryCapDataBlock` function may need to be called prior to call this function.

#### Format

```
Call PD_GetBatteryCap()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call PD_GetBatteryCap()
```

### 5.1.68 PD_SetGetBatteryCapDataBlock

Applicable to PD Rev 3.0 only. Sets the *GetBatteryCap Data Block* in PD Exerciser. It has to be called prior calling the PD_GetBatteryCap function to take effect.

**Format**

```
Call PD_SetGetBatteryCapDataBlock( PD_GetBatteryCapDataBlock $get_battery_cap_db )
```

**Parameters**

`$get_battery_cap_db`

Parameter type is PD_GetBatteryCapDataBlock. Refer to PD_GetBatteryCapMsg for available fields of this type.

**Result**

None

**Examples**

```
$get_battery_cap_db = PD_GetBatteryCapDataBlock
Call PD_SetGetBatteryCapDataBlock( $get_battery_cap_db )
```

### 5.1.69 PD_SetGetBatteryCapSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the PD_WaitForGetBatteryCap or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetGetBatteryCapSetting( PD_GetBatteryCap_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is PD_GetBatteryCap_Settings. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(micro second) to receive Get_Battery_Cap message. |
| ResponseType | PD_RESPONSE_NOT_SUPPORTED, PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the Get_Battery_Cap message. |

**Result**

None

**Examples**

```
$getbattcap_setting = PD_GetBatteryCap_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call PD_SetGetBatteryCapSetting( $getbattcap_setting )
```

### 5.1.70 PD_WaitForGetBatteryCap

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Get_Battery_Cap` message. It will respond to incoming messages as part of the *GetBatteryCap* AMS.

**Note:** The `PD_SetGetBatteryCapSetting`, `PD_SetBatteryCapDataBlock` and `PD_ResetBatteryCapDataBlock` functions may need to be called prior to call this function.

#### Format

```
Call PD_WaitForGetBatteryCap()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Battery_Cap` not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

#### Examples

```
Call PD_WaitForGetBatteryCap()
```

### 5.1.71 PD_SetBatteryCapDataBlock

Applicable to PD Rev 3.0 only. Sets the *BatteryCap Data Block* in PD Exerciser. It has to be called prior calling the `PD_WaitForGetBatteryCap` or `PD_DelayAutoResponse` functions to take effect.

#### Format

```
Call PD_SetBatteryCapDataBlock( PD_BatteryCapDataBlock $battery_cap_db )
```

#### Parameters

`$battery_cap_db`

Parameter type is `PD_BatteryCapDataBlock`. Refer to `PD_BatteryCapabilitiesMsg` for available fields of this type.

#### Result

None

#### Examples

```
$battery_cap_db = PD_BatteryCapDataBlock
Call PD_SetBatteryCapDataBlock( $battery_cap_db )
```

### 5.1.72 PD_ResetBatteryCapDataBlock

Applicable to PD Rev 3.0 only. Clears the *BatteryCap Data Block* in PD Exerciser. Should be called before calling `PD_SetBatteryCapDataBlock` command.

**Format**

```
Call PD_ResetBatteryCapDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call PD_ResetBatteryCapDataBlock()
```

### 5.1.73 PD_GetManufacturerInfo

Applicable to PD Rev 3.0 only. Starts *GetManufacturerInfo* AMS.

**Note:** The `PD_SetGetManufacturerInfoDataBlock` function may need to be called prior to call this function.

**Format**

```
Call PD_GetManufacturerInfo( OrderedSetType )
```

**Parameters**

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call PD_GetManufacturerInfo( PD_ORDERED_SET_TYPE_SOP )
```

### 5.1.74 PD_SetGetManufacturerInfoDataBlock

Applicable to PD Rev 3.0 only. Sets *GetManufacturerInfo Data Block* in PD Exerciser. It has to be called prior calling the `PD_GetManufacturerInfo` function to take effect.

**Format**

```
Call PD_SetGetManufacturerInfoDataBlock( PD_GetManufacturerInfoDataBlock $get_manuf_info_db )
```

**Parameters**

`$get_manuf_info_db`

Parameter type is `PD_GetManufacturerInfoDataBlock`. Refer to

`PD`_GetManufacturerInfoMsg for available fields of this type.

**Result**

None

**Examples**

```
$get_manuf_info_db = PD_GetManufacturerInfoDataBlock
Call PD_SetGetManufacturerInfoDataBlock( $get_manuf_info_db )
```

### 5.1.75 PD_SetGetManufacturerInfoSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForGetManufacturerInfo` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetGetManufacturerInfoSetting( PD_GetManufacturerInfo_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is PD_GetManufacturerInfo_Settings. Available fields of this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(in micro seconds) to receive `Get_Manufacturer_Info` message. |
| ResponseType | `PD_RESPONSE_NOT_SUPPORTED`, `PD_RESPONSE_UNSPECIFIED`(default) | Indicates response upon receiving the `Get_Manufacturer_Info` message. |

**Result**

None

**Examples**

```
$getmaninfo_setting = PD_GetManufacturerInfo_Settings
{
   WaitTimeout = 50000
}
Call PD_SetGetManufacturerInfoSetting( $getmaninfo_setting )
```

### 5.1.76 PD_WaitForGetManufacturerInfo

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Manufacturer_Info` message. It will respond to incoming messages as part of the *GetManufacturerInfo* AMS.

**Note:** The `PD_SetGetManufacturerInfoSetting` and `PD_SetManufacturerInfoDataBlock` functions may need to be called prior to call this function.

**Format**

```
Call PD_WaitForGetManufacturerInfo()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Manufacturer_Info` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call PD_WaitForGetManufacturerInfo()
```

### 5.1.77 PD_SetManufacturerInfoDataBlock

Applicable to PD Rev 3.0 only. Sets *ManufacurerInfo Data Block* in PD Exerciser. It has to be called prior calling the `PD_WaitForGetManufacturerInfo` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetManufacturerInfoDataBlock( PD_ManufacturerInfoDataBlock $manufacturer_info_db )
```

**Parameters**

`$manufacturer_info_db`

Parameter type is `PD_ManufacturerInfoDataBlock`. Refer to `PD_ManufacturerInfoMsg` for available fields of this type.

**Result**

None

**Examples**

```
$manufacturer_info_db = PD_ManufacturerInfoDataBlock
Call PD_SetManufacturerInfoDataBlock( $manufacturer_info_db )
```

### 5.1.78 PD_SetSecurityRequestSetting

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForSecurityRequest` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetSecurityRequestSetting( PD_SecurityRequest_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is `PD_SecurityRequest_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(in micro seconds) to receive Security_Request message. |
| ResponseType | PD_RESPONSE_NOT_SUPPORTED, PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the Security_Request message. |

**Result**

None

**Examples**

```
$secreq_settings = PD_SecurityRequest_Settings
{
   WaitTimeout = 50000
}
Call PD_SetSecurityRequestSetting( $secreq_settings )
```

### 5.1.79 PD_SecurityRequest

Applicable to PD Rev 3.0 only. Starts the *SecurityRequest* AMS.

**Note:** The PD_SetSecurityRequestDataBlock function may need to be called prior to call this function.

**Format**

```
Call PD_SecurityRequest( OrderedSetType )
```

**Parameters**

OrderedSetType

Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message received. |

**Examples**

```
Call PD_SecurityRequest( PD_ORDERED_SET_TYPE_SOP )
```

### 5.1.80 PD_SetSecurityRequestDataBlock

Applicable to PD Rev 3.0 only. Sets the *SecurityRequest Data Block* in PD Exerciser. It has to be called prior calling the PD_SecurityRequest function to take effect.

**Format**

```
Call PD_SetSecurityRequestDataBlock( PD_SecurityRequestDB $security_req_db )
```

### Parameters

`$security_req_db`

Parameter type is PD_SecurityRequestDB. Refer to `PD_SecurityRequestMsg` for available types which are derived from this type.

### Result

None

### Examples

```
$security_req_db = PD_SRQDB_GetDigests
Call PD_SetSecurityRequestDataBlock( $security_req_db )
```

## 5.1.81 PD_WaitForSecurityRequest

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Security_Request` message. It will respond to incoming messages as part of the *SecurityRequest* AMS.

**Note:** The `PD_SetSecurityRequestSetting` and `PD_SetSecurityResponseDataBlock` functions may need to be called prior to call this function.

### Format

```
Call PD_WaitForSecurityRequest()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Security_Request` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

### Examples

```
Call PD_WaitForSecurityRequest()
```

## 5.1.82 PD_SetSecurityResponseDataBlock

Sets the SecurityResponse Data Block in PD Exerciser. It must be called before `PD_WaitForSecurityRequest` or `PD_DelayAutoResponse` to take effect.

### Format

```
Call PD_SetSecurityResponseDataBlock( PD_SecurityResponseDB $security_resp_db )
```

### Parameters

`$security_resp_db`

Parameter type is `PD_SecurityResponseDB`. Refer to `PD_SecurityResponseMsg` for available types which are derived from this type.

**Result**

None

**Examples**

```
$security_resp_db = PD_SRPDB_Certificate
Call PD_SetSecurityResponseDataBlock( $security_resp_db )
```

### 5.1.83 Pd_SetGetCountryInfoSetting

Applicable to PD Rev 3.0 only. It has to be called prior to call `Pd_WaitForGetCountryInfo` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call Pd_SetGetCountryInfoSetting( Pd_GetCountryInfo_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `Pd_GetCountryInfo_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(micro second) to receive `Get_Country_Info` message. |
| ResponseType | `PD_RESPONSE_NOT_SUPPORTED`, `PD_RESPONSE_UNSPECIFIED`(default) | Indicates response upon receiving the `Get_Country_Info` message. |

**Result**

None

**Examples**

```
$getcountryinfo_setting = Pd_GetCountryInfo_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call Pd_SetGetCountryInfoSetting( $getcountryinfo_setting )
```

### 5.1.84 Pd_SetCountryInfoDataBlock

Applicable to PD Rev 3.0 only. Sets *CountryInfo Data Block* in PD Exerciser. It has to be called prior to call `Pd_WaitForGetCountryInfo` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call Pd_SetCountryInfoDataBlock( Pd_CountryInfoDataBlock $country_info_db )
```

**Parameters**

`$country_info_db`

parameter type is `Pd_CountryInfoDataBlock`. Refer to `PD_CountryInfoMsg` for available data fields.

**Result**

None

**Examples**

```
$country_info_db = Pd_CountryInfoDataBlock
Call Pd_SetCountryInfoDataBlock( $country_info_db )
```

### 5.1.85 Pd_ResetCountryInfoDataBlock

Applicable to PD Rev 3.0 only. Clears the *CountryInfo Data Block* in PD Exerciser. Should be called prior to call `Pd_SetCountryInfoDataBlock`.

**Format**

```
Call Pd_ResetCountryInfoDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call Pd_ResetCountryInfoDataBlock()
```

### 5.1.86 Pd_SetGetCountryInfoDO

Applicable to PD Rev 3.0 only. Sets *GetCountryInfo Data Object* in PD Exerciser. It has to be called prior to call `Pd_GetCountryInfo` function to take effect.

**Format**

```
Call Pd_SetGetCountryInfoDO( Pd_GetCountryInfoDO $get_country_info_do )
```

**Parameters**

`$get_country_info_do`

parameter type is `Pd_GetCountryInfoDO`. Refer to `PD_GetCountryInfoMsg` for available data fields.

**Result**

None

**Examples**

```
$get_country_info_do = Pd_GetCountryInfoDO
Call Pd_SetGetCountryInfoDO( $get_country_info_do )
```

### 5.1.87 Pd_GetCountryInfo

Applicable to PD Rev 3.0 only. Starts *GetCountryInfo* AMS.

Note: The `Pd_SetGetCountryInfoDO` function may need to be called prior calling this function.

**Format**

```
Call Pd_GetCountryInfo( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call Pd_GetCountryInfo(PD_ORDERED_SET_TYPE_SOP)
```

## 5.1.88 Pd_WaitForGetCountryInfo

Applicable to PD Rev 3.0 only. Wait for user-defined timeout to receive `Get_Country_Info` message. It will respond to incoming messages as part of *GetCountryInfo* AMS.

**Note:** The `Pd_SetGetCountryInfoSetting`, `Pd_SetCountryInfoDataBlock` and `Pd_ResetCountryInfoDataBlock` functions may need to be called prior calling this function.

**Format**

```
Call Pd_WaitForGetCountryInfo()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Country_info` not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call Pd_WaitForGetCountryInfo()
```

## 5.1.89 Pd_SetGetCountryCodesSetting

Applicable to PD Rev 3.0 only. It has to be called prior to call `Pd_WaitForGetCountryCodes` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call Pd_SetGetCountryCodesSetting( Pd_GetCountryCodes_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `Pd_GetCountryCodes_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait TimeOut(micro second) to receive `Get_Country_Codes` message. |
| ResponseType | `PD_RESPONSE_NOT_SUPPORTED`, `PD_RESPONSE_UNSPECIFIED`(default) | Indicates response upon receiving the `Get_Country_Codes` message. |

**Result**

None

**Examples**

```
$getcountrycodes_setting = Pd_GetCountryCodes_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call Pd_SetGetCountryCodesSetting( $getcountrycodes_setting )
```

### 5.1.90 Pd_SetCountryCodesDataBlock

Applicable to PD Rev 3.0 only. Sets *CountryCodes Data Block* in PD Exerciser. It has to be called prior to call `Pd_WaitForGetCountryCodes` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call Pd_SetCountryCodesDataBlock( Pd_CountryCodesDataBlock $country_codes_db )
```

**Parameters**

`$country_codes_db`

parameter type is `Pd_CountryCodesDataBlock`. Refer to `PD_CountryCodesMsg` for available data fields.

**Result**

None

**Examples**

```
$country_codes_db = Pd_CountryCodesDataBlock
Call Pd_SetCountryCodesDataBlock( $country_codes_db )
```

### 5.1.91 Pd_ResetCountryCodesDataBlock

Applicable to PD Rev 3.0 only. Clears the *CountryCodes Data Block* in PD Exerciser. Should be called prior to call `Pd_SetCountryCodesDataBlock`.

**Format**

```
Call Pd_ResetCountryCodesDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call Pd_ResetCountryCodesDataBlock()
```

### 5.1.92 Pd_GetCountryCodes

Applicable to PD Rev 3.0 only. Starts *GetCountryCodes* AMS.

**Format**

```
Call Pd_GetCountryCodes( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call Pd_GetCountryCodes(PD_ORDERED_SET_TYPE_SOP)
```

### 5.1.93 Pd_WaitForGetCountryCodes

Applicable to PD Rev 3.0 only. Wait for user-defined timeout to receive `Get_Country_Codes` message. It will respond to incoming messages as part of *GetCountryCodes* AMS.

**Note:** The `Pd_SetGetCountryCodesSetting`, `Pd_SetCountryCodesDataBlock` and `Pd_ResetCountryCodesDataBlock` functions may need to be called prior calling this function.

**Format**

```
Call Pd_WaitForGetCountryCodes()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Country_Codes` not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call Pd_WaitForGetCountryCodes()
```

### 5.1.94 PD_EnterUSB

Applicable to PD Rev 3.0 only. Starts *EnterUSB* AMS.

**Format**

```
Call Pd_EnterUSB( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as `Request` message response |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call Pd_EnterUSB(PD_ORDERED_SET_TYPE_SOP)
```

### 5.1.95 PD_WaitForEnterUSB

Applicable to PD Rev 3.0 only. Wait for user-defined timeout to receive `EnterUSB` message. It will respond to incoming messages as part of *EnterUSB* AMS.

Note: The `PD_SetEnterUSBSetting` function may need to be called prior calling this function.

### Format

```
Call Pd_WaitForEnterUSB()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

### Examples

```
Call Pd_WaitForEnterUSB()
```

## 5.1.96 PD_SetEnterUSBSetting

Applicable to PD Rev 3.0 only. It has to be called prior to call **PD_WaitForEnterUSB** or **PD_DelayAutoResponse** functions to take effect.

### Format

```
Call Pd_SetEnterUSBSetting( Pd_EnterUSB_Settings $settings )
```

### Parameters

`$settings`

The setting type is `Pd_EnterUSB_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait timeout (microsecond) to receive `EnterUSB` message. |
| ResponseType | `PD_RESPONSE_REJECT`,<br>`PD_RESPONSE_WAIT`,<br>`PD_RESPONSE_ACCEPT`(default) | Indicates response upon receiving the `EnterUSB` message. |
| AcceptedUSBMode | `PD_INVALID_VALUE` (default),<br>`PD_USBMODE_USB2`,<br>`PD_USBMODE_USB3`,<br>`PD_USBMODE_USB4` | When the value is not PD_INVALID_VALUE, Exerciser sends a Reject response if it doesn't match the `USBMode` field in the received Enter_USB message. |
| AcceptedCableSpeed | `PD_INVALID_VALUE` (default),<br>`PD_CABLESPEED_USB2_NO_SSP`,<br>`PD_CABLESPEED_USB32_GEN1`,<br>`PD_CABLESPEED_USB32_USB4_GEN2`,<br>`PD_CABLESPEED_USB4_GEN3`,<br>`PD_CABLESPEED_USB4_GEN4` | When the value is not PD_INVALID_VALUE, Exerciser sends a Reject response if it doesn't match the `CableSpeed` field in the received Enter_USB message. |
| AcceptedCableSpeedHigherEqual | `PD_INVALID_VALUE` (default),<br>`PD_CABLESPEED_USB2_NO_SSP`,<br>`PD_CABLESPEED_USB32_GEN1`,<br>`PD_CABLESPEED_USB32_USB4_GEN2`,<br>`PD_CABLESPEED_USB4_GEN3`,<br>`PD_CABLESPEED_USB4_GEN4` | When the value is not PD_INVALID_VALUE, the Exerciser sends a Reject response if it is lower than the `CableSpeed` field in the received Enter_USB message. |

| | | |
|---|---|---|
| AcceptedCableSpeedLowerEqual | PD_INVALID_VALUE (default),<br>PD_CABLESPEED_USB2_NO_SSP,<br>PD_CABLESPEED_USB32_GEN1,<br>PD_CABLESPEED_USB32_USB4_GEN2,<br>PD_CABLESPEED_USB4_GEN3,<br>PD_CABLESPEED_USB4_GEN4 | When the value is not PD_INVALID_VALUE, the Exerciser sends a Reject response if it is higher than the CableSpeed field in the received Enter_USB message. |
| AcceptedCableType | PD_INVALID_VALUE (default),<br>PD_CABLETYPE_PASSIVE,<br>PD_CABLETYPE_ACTIVE_RETIMER,<br>PD_CABLETYPE_ACTIVE_REDRIVER,<br>PD_CABLETYPE_OPTICALLY_ISOLATED | When the value is not PD_INVALID_VALUE, Exerciser sends a Reject response if it doesn't match the CableType field in the received Enter_USB message. |
| AcceptedCableCurrent | PD_INVALID_VALUE (default),<br>PD_CABLECURRENT_VBUS_NOT_SUPP,<br>PD_CABLECURRENT_3A,<br>PD_CABLECURRENT_5A | When the value is not PD_INVALID_VALUE, Exerciser sends a Reject response if it doesn't match the CableCurrent field in the received Enter_USB message. |
| RejectNonZeroReserved | PD_TRUE,<br>PD_FALSE (default) | If the value is PD_TRUE, Exerciser sends a Reject response in case of non-zero reserved fields in the received Enter_USB message. |
| RetryCountOnWait | 2 (default) | Indicates the retry count if Wait message received as response. |
| RetryDelayOnWait | 100000 (default) | Indicates the retry delay time(micro second) if Wait message received as response. |

**Result**

None

**Examples**

```
$enterusb_setting = Pd_EnterUSB_Settings
{
   ResponseType = PD_RESPONSE_REJECT
}
Call Pd_SetEnterUSBSetting( $enterusb_setting )
```

### 5.1.97 PD_SetEnterUSBDO

Applicable to PD Rev 3.0 only. Sets *PD_EnterUSBDataObject* in PD Exerciser. It has to be called prior to call PD_EnterUSB function to take effect.

**Format**

```
Call Pd_SetEnterUSBDO( Pd_EnterUSBDataObject $enter_usb_do )
```

**Parameters**

$enter_usb_do

Parameter type is Pd_EnterUSBDataObject. Refer to PD_EnterUSBDataObject for available data fields.

**Result**

None

**Examples**

```
$enter_usb_do = Pd_EnterUSBDataObject
Call Pd_EnterUSBDO( $enter_usb_do )
```

### 5.1.98 PD_ResetEnterUSBDO

Applicable to PD Rev 3.0 only. Clears the *Pd_EnterUSBDataObject* in PD Exerciser. Should be called prior to call PD_SetEnterUSBDO.

**Format**

```
Call Pd_ResetEnterUSBDO()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call Pd_ResetEnterUSBDO()
```

### 5.1.99 PD_DiscoverAndEnterUSB4

Starts the process of USB4 discovery and entry.

Use `PD_SetDiscoverAndEnterUSB4Settings()` command to configure the command.

**Format**

```
Call PD_DiscoverAndEnterUSB4( $enter_usb_do )
```

**Parameters**

`$enter_usb_do`

Parameter type is `Pd_EnterUSBDataObject`. Refer to `PD_EnterUSBDataObject` for available data fields.

**Result**

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. |

**Examples**

```
Call PD_DiscoverAndEnterUSB4() # It can be called without any parameter also
```

### 5.1.100 PD_SetDiscoverAndEnterUSB4Settings

It used to configure the `PD_DiscoverAndEnterUSB4()` command.

**Format**

```
Call Pd_SetDiscoverAndEnterUSB4Settings( Pd_DiscoverAndEnterUSB4_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `Pd_DiscoverAndEnterUSB4_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| SkipSOP | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to skip discovery and entry of SOP. |
| SkipCable | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to skip discovery and entry of SOP' and SOP". |
| AttemptCable | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to attempt discovery and entry of SOP' and SOP" even if it is not an Active cable. |
| USBMode | PD_USBMODE_USB2<br>PD_USBMODE_USB3<br>PD_USBMODE_USB4(default) | Ignored when Enter_USB Data Object is passed to PD_DiscoverAndEnterUSB4() command. |
| ExrHighestSpeed | PD_USB_HIGHEST_SPEED_USB20_ONLY<br>PD_USB_HIGHEST_SPEED_USB32_GEN1<br>PD_USB_HIGHEST_SPEED_USB32_USB4_GEN2<br>PD_USB_HIGHEST_SPEED_USB4_GEN3(default)<br>PD_USB_HIGHEST_SPEED_USB4_GEN4 | |
| SkipSpeedCheck | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to skip speed test check. it results in sending an Enter_USB message regardless of the speed |
| SkipTBT3ActiveCableCheck | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to skip the tbt3 active cable check when the highest speed is USB3_USB4_Gen2 |

### Result

None

### Examples

```
$discovery_setting = Pd_DiscoverAndEnterUSB4_Settings
{
   USBMode = PD_USBMODE_USB3
}
Call PD_SetDiscoverAndEnterUSB4Settings( $discovery_setting )
```

## 5.1.101 Pd_SetGetSnkCapExtSetting

Applicable to PD Rev 3.0 only. It has to be called before Pd_GetSinkCapExtended or Pd_WaitForGetSnkCapExtended or PD_DelayAutoResponse functions to take effect.

### Format

```
Call Pd_SetGetSnkCapExtSetting( Pd_GetSinkCapExtented_Settings $settings )
```

### Parameters

$settings

Setting type is Pd_GetSinkCapExtented_Settings. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait TimeOut(micro second) to receive Get_Sink_Cap_Extended message. |
| ResponseType | PD_RESPONSE_NOT_SUPPORTED,<br>PD_RESPONSE_UNSPECIFIED(default) | Indicates response upon receiving the Get_Sink_Cap_Extended message. |
| SkipSnkCapExt | PD_TRUE,<br>PD_FALSE(default) | Indicates whether to skip sending Sink_Capabilities_Extended message or not. |
| SendSnkCapExtDelay | 0(default) | Defines the delay before sending Sink_Capabilities_Extended message. |

### Result

None

**Examples**

```
$getsnkcapext_setting = Pd_GetSinkCapExtented_Settings
{
   ResponseType = PD_RESPONSE_NOT_SUPPORTED
}
Call Pd_SetGetSnkCapExtSetting( $getsnkcapext_setting )
```

### 5.1.102 Pd_SetSnkCapExtDataBlock

Applicable to PD Rev 3.0 only. Sets *Sink Capabilities Extended Data Block* in PD Exerciser. It has to be called before `Pd_WaitForGetSnkCapExtended` or `PD_DelayAutoResponse` to take effect.

**Format**

```
Call Pd_SetSnkCapExtDataBlock( Pd_SinkCapExtDataBlock $snk_cap_ext )
```

**Parameters**

`$snk_cap_ext`

parameter type is `Pd_SinkCapExtDataBlock`. Refer to `PD_SinkCapExtendedMsg` for available data fields.

**Result**

None

**Examples**

```
$snk_cap_ext = Pd_SinkCapExtDataBlock
Call Pd_SetSnkCapExtDataBlock( $snk_cap_ext )
```

### 5.1.103 Pd_ResetSnkCapExtDataBlock

Applicable to PD Rev 3.0 only. Clears the *Sink Capabilities Extended Data Block* in PD Exerciser. Should be called prior to call `Pd_SetSnkCapExtDataBlock`.

**Format**

```
Call Pd_ResetSnkCapExtDataBlock()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call Pd_ResetSnkCapExtDataBlock()
```

### 5.1.104 Pd_GetSinkCapExtended

Applicable to PD Rev 3.0 only. Starts *GetSinkCapExtended* AMS.

**Note:** The `Pd_SetGetSnkCapExtSetting` function may need to be called prior calling this function.

**Format**

```
Call Pd_GetSinkCapExtended()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received. |

**Examples**

```
Call Pd_GetSinkCapExtended()
```

### 5.1.105 Pd_WaitForGetSnkCapExtended

Applicable to PD Rev 3.0 only. Wait for user-defined timeout to receive `Get_Sink_Cap_Extended` message. It will respond to incoming messages as part of *GetSinkCapExtended* AMS.

**Note:** The `Pd_SetGetSnkCapExtSetting`, `Pd_SetSnkCapExtDataBlock` and `Pd_ResetSnkCapExtDataBlock` functions may need to be called prior calling this function.

**Format**

```
Call Pd_WaitForGetSnkCapExtended()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Sink_Cap_Extended` not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message sent as response. |

**Examples**

```
Call Pd_WaitForGetSnkCapExtended()
```

### 5.1.106 PD_SetDiscoverIdentitySetting

It must be called prior calling the `PD_DiscoverIdentity` or `PD_WaitForDiscoverIdentity` or `PD_PerformDiscoveryProcess` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetDiscoverIdentitySetting( PD_DiscoverIdentity_Settings $settings )
```

### Parameters

`$settings`

Should be from `PD_DiscoverIdentity_Settings` type. Table below shows the available fields of `PD_DiscoverIdentity_Settings` template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| DiscoverIdentityRetryCount | 20 (default) | Indicates the DiscoverIdentity retry count. |
| DiscoverIdentityResponse | PD_DISCOVERIDENTITY_ACK(default)<br>PD_DISCOVERIDENTITY_BUSY<br>PD_DISCOVERIDENTITY_NAK<br>PD_RESPONSE_NOT_SUPPORTED | Indicates the response type. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Timeout(micro second) to wait for receiving Discover Identity Command. |
| RetryCountOnWait | 4(default) | Indicates the retry count if `Wait` message received as response. |
| RetryDelayOnWait | 50000(default) | Indicates the retry delay time(micro second) if `Wait` message received as response. |
| AutoSpecRevCable | PD_TRUE(default) | Only applicable for Rev.3.0 or higher. Indicates whether to detect Specification Revision of messages towards the cable automatically or not. |
| SendResponseDelay | 0(default) | Sends response after specified delay. |

### Result

None

### Examples

```
#apply settings
$settings = PD_DiscoverIdentity_Settings
{
   DiscoverIdentityRetryCount = 5
}
call PD_SetDiscoverIdentitySetting( $settings )
```

## 5.1.107 PD_AddDiscoverIdentityVDO

Adds *DiscoverIdentity VDO* in PD Exerciser. It has to be called prior calling the `PD_WaitForDiscoverIdentity` or `PD_DelayAutoResponse` functions to take effect.

### Format

```
Call PD_AddDiscoverIdentityVDO( PD_DiscoverIdentity_VDO $vdo )
```

### Parameters

`$vdo`

Parameter type is PD_DiscoverIdentity_VDO. Refer to `PD_VDM_Discover_Identity_Response` for available DiscoverID VDOs.

### Result

None

### Examples

```
#In this example, PD working revision is PD_SPEC_REVISION_2
#Add a ID Header VDO
$vdo = PD_VDM_Discover_Identity_ID_Header_VDO
{
   IDHeaderVDO_USBVendorID = 0xFF01
   IDHeaderVDO_ModalOperationSupported = 1
   IDHeaderVDO_ProductType = PD_VDM_ID_HEADER_VDO_PRODUCT_TYPE_PERIPHERAL
   IDHeaderVDO_DataCapableAsUSBDevice = 1
}
call PD_AddDiscoverIdentityVDO( $vdo )
```

### 5.1.108 PD_ResetDiscoverIdentityVDO

Clears *DiscoverIdentity VDOs* in PD Exerciser. It should be called prior calling the `PD_AddDiscoverIdentityVDO` function.

**Format**

```
Call PD_ResetDiscoverIdentityVDO()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetDiscoverIdentityVDO()
```

### 5.1.109 PD_DiscoverIdentity

Starts DiscoverIdentity AMS.

**Format**

```
Call PD_DiscoverIdentity( OrderedSetType )
```

**Parameters**

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - `NAK` received as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - `BUSY` received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

**Examples**

```
call PD_DiscoverIdentity(PD_ORDERED_SET_TYPE_SOP)
```

### 5.1.110 PD_WaitForDiscoverIdentity

Waits for user-defined timeout to receive DISCOVERIDENTITY command. It will respond to incoming messages as part of *DiscoverIdentity* AMS.

**Note:** The PD_SetDiscoverIdentitySetting, PD_AddDiscoverIdentityVDO and PD_ResetDiscoverIdentityVDO functions may need to be called prior calling this function.

### Format

```
Call PD_WaitForDiscoverIdentity()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVERIDENTITY command not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

### Examples

```
call PD_WaitForDiscoverIdentity()
```

## 5.1.111 PD_SetDiscoverSVIDSetting

It has to be called prior calling the PD_DiscoverSvids or PD_WaitForDiscoverSvids or PD_PerformDiscoveryProcess or PD_DelayAutoResponse functions to take effect.

### Format

```
Call PD_SetDiscoverSVIDSetting( PD_DiscoverSvids_Settings $settings )
```

### Parameters

$settings

Should be from PD_DiscoverSvids_Settings type. Table below shows the available fields of PD_DiscoverSvids_Settings template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| DiscoverSvidsResponse | PD_DISCOVERSVIDS_ACK(default)<br>PD_DISCOVERSVIDS_BUSY<br>PD_DISCOVERSVIDS_NAK<br>PD_RESPONSE_NOT_SUPPORTED | Indicates the response type. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Timeout(micro second) to wait for receiving Discover SVID Command. |
| RetryCountOnWait | 4(default) | Indicates the retry count if Wait message received as response. |
| RetryDelayOnWait | 50000(default) | Indicates the retry delay time(micro second) if Wait message received as response. |

### Result

None

### Examples

```
#Using default settings
$settings = PD_DiscoverSvids_Settings
```

```
call PD_SetDiscoverSVIDSetting( $settings )
```

### 5.1.112 PD_AddSvid

Adds *SVIDs* to PD Exerciser. It has to be called prior calling the PD_DiscoverSvids or PD_WaitForDiscoverSvids or PD_PerformDiscoveryProcess or PD_DelayAutoResponse functions to take effect.

**Note** - Up to 11 *SVIDs* can be added using this command.

**Format**

```
Call PD_AddSvid(value)
```

**Parameters**

`value`

SVID value to add

**Result**

None

**Examples**

```
call PD_AddSvid(0xFF01)
```

### 5.1.113 PD_ResetSvids

Clears *SVIDs* which is added to PD Exerciser. It should be called prior calling the PD_AddSvid function.

**Format**

```
Call PD_ResetSvids()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetSvids()
```

### 5.1.114 PD_DiscoverSvids

Starts *DiscoverSVID* AMS.

**Note** - PD Exerciser supports only one(first) DiscoverSVIDs Ack message.

**Format**

```
Call PD_DiscoverSvids(OrderedSetType)
```

**Parameters**

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
```

PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME

### Result

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK received as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message received as response |

### Examples

```
call PD_DiscoverSvids(PD_ORDERED_SET_TYPE_SOP)
```

## 5.1.115 PD_WaitForDiscoverSvids

Waits for user-defined timeout to receive DISCOVERSVID command. It will respond to incoming messages as part of the *DiscoverSVIDs* AMS.

**Note:** The PD_SetDiscoverSVIDSetting, PD_AddSvid and PD_ResetSvids functions may need to be called prior calling this function.

### Format

```
Call PD_WaitForDiscoverSvids()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVERSVIDS message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

### Examples

```
call PD_WaitForDiscoverSvids()
```

## 5.1.116 PD_SetDiscoverModeSetting

It has to be called prior calling the PD_DiscoverModes or PD_WaitForDiscoverModes or PD_PerformDiscoveryProcess or PD_DelayAutoResponse functions to take effect.

### Format

```
Call PD_SetDiscoverModeSetting( PD_DiscoverModes_Settings $settings )
```

### Parameters

`$settings`

It should be from `PD_DiscoverModes_Settings` type. Table below describes the `PD_DiscoverModes_Settings` template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| DiscoverModesResponse | `PD_DISCOVERMODES_ACK`(default)<br>`PD_DISCOVERMODES_BUSY`<br>`PD_DISCOVERMODES_NAK`<br>`PD_RESPONSE_NOT_SUPPORTED` | Response type |
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Timeout(micro second) to wait for receiving Discover Modes command. |
| RetryCountOnWait | 4(default) | Indicates the retry count if `Wait` message received as response. |
| RetryDelayOnWait | 50000(default) | Indicates the retry delay time(micro second) if `Wait` message received as response. |

### Result

None

### Examples

```
#apply settings
$settings = PD_DiscoverModes_Settings
{
   DiscoverModesResponse = PD_DISCOVERMODES_BUSY
}
call PD_SetDiscoverModeSetting( $settings )
```

## 5.1.117 PD_AddMode

Adds *Modes* to the PD Exerciser. It has to be called prior calling the `PD_DiscoverModes` or `PD_WaitForDiscoverModes` or `PD_PerformDiscoveryProcess` or `PD_DelayAutoResponse` functions to take effect.

### Format

```
Call PD_AddMode(Mode)
```

### Parameters

`Mode`

Mode to add

### Result

None

### Examples

```
call PD_AddMode(0x00000001)
```

## 5.1.118 PD_AddModeVDO

Adds *Mode with VDO* to the PD Exerciser. It has to be called prior calling the `PD_DiscoverModes` or `PD_WaitForDiscoverModes` or `PD_PerformDiscoveryProcess` or `PD_DelayAutoResponse` functions to take effect.

### Format

```
Call PD_AddModeVDO(PD_Generic_VDO $ModeVdo)
```

### Parameters

`$ModeVdo`

Parameter type is `PD_Generic_VDO`. Refer to `PD_VDM_Discover_Modes_Response` for available VDOs which can be use as this parameter.

**Result**

None

**Examples**

```
local $vdo_1 = PD_VDO
{
   Data = 0x01
}
call PD_AddModeVDO($vdo_1)
```

### 5.1.119 PD_ResetModes

Clears `Modes` which are added to PD Exerciser. It should be called prior calling the `PD_AddMode` or `PD_AddModeVDO` functions.

**Format**

```
Call PD_ResetModes()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetModes()
```

### 5.1.120 PD_DiscoverModes

Starts *DicoverModes* AMS.

**Format**

```
Call PD_DiscoverModes(OrderedSetType, selectedSvid)
```

**Parameters**

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID value

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|

| | |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK received as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message received as response |

**Examples**

```
call PD_DiscoverModes(PD_ORDERED_SET_TYPE_SOP,0xFF00)
```

### 5.1.121 PD_WaitForDiscoverModes

Waits for user-defined timeout to receive DISCOVERMODE command. It will respond to incoming messages as part of *DiscoverModes* AMS.

**Note:** The PD_SetDiscoverModeSetting, PD_AddMode, PD_AddModeVDO and PD_ResetModes functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForDiscoverModes()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVERMODES message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

**Examples**

```
call PD_WaitForDiscoverModes()
```

### 5.1.122 PD_SetEnterModeSetting

It has to be called prior calling the PD_WaitForEnterMode or PD_DelayAutoResponse functions to take effect.

**Format**

```
Call PD_SetEnterModeSetting( PD_EnterMode_Settings $settings )
```

**Parameters**

`$settings`

Should be from PD_EnterMode_Settings type. Table below describes the PD_EnterMode_Settings template:

| Field Name | Possible/Default Values | Description |
|---|---|---|

| | | |
|---|---|---|
| EnterModeResponse | PD_ENTERMODE_ACK(default)<br>PD_ENTERMODE_NAK<br>PD_RESPONSE_NOT_SUPPORTED | Response type. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Timeout(micro second) to wait for receiving Enter Mode command. |
| SkipSendingResponse | PD_FALSE(default)<br>PD_TRUE | If set to PD_TRUE, will skip sending the EnterMode response. |

**Result**

None

**Examples**

```
#Using default setting
$settings = PD_EnterMode_Settings
call PD_SetEnterModeSetting( $settings )
```

### 5.1.123 PD_EnterMode

Starts *EnterMode* AMS.

**Format**

```
Call PD_EnterMode(OrderedSetType, selectedSvid, modeIndex)
```

**Parameters**

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID

`modeIndex`

Indicates the mode index for the specified SVID

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - Not_Supported message received as response |

**Examples**

```
call PD_EnterMode(PD_ORDERED_SET_TYPE_SOP,0xFF00, 1)
```

### 5.1.124 PD_EnterModeVdo

Starts *EnterMode* AMS. It sends the specified *VDO* using the *EnterMode* command.

#### Format

```
Call PD_EnterModeVdo( OrderedSetType, selectedSvid, modeId, PD_Generic_VDO $Vdo )
```

#### Parameters

`OrderedSetType`

Indicates the ordered set type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID

`modeId`

Indicates the mode index related to the specified SVID

`$Vdo`

Vendor defined data object. It should be from `PD_VDO`(Inherited from `PD_Generic_VDO`) type. (Refer to `PD_VDO`)

| Field Name | Description |
|---|---|
| Data | VDO data |

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK received as response |

#### Examples

```
$vdo = PD_VDO
{
   Data = 0x00
}
call PD_EnterModeVdo(PD_ORDERED_SET_TYPE_SOP, 0xFF01, 1, $vdo)
```

### 5.1.125 PD_WaitForEnterMode

Waits for user-defined timeout to receive ENTERMODE command. It will respond to incoming messages as part of *EnterMode* AMS.

**Note:** The `PD_SetEnterModeSetting` function may need to be called prior calling this function.

#### Format

```
Call PD_WaitForEnterMode()
```

### Parameters

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - ENERMODE message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |

### Examples

```
call PD_WaitForEnterMode()
```

## 5.1.126 PD_SetExitModeSetting

It has to be called prior calling the **PD_WaitForExitMode** or **PD_DelayAutoResponse** functions to take effect.

### Format

```
Call PD_SetExitModeSetting( PD_ExitMode_Settings $settings )
```

### Parameters

`$settings`

Should be from `PD_ExitMode_Settings` type. Table below describes the `PD_ExitMode_Settings` template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| ExitModeResponse | PD_EXITMODE_ACK(default)<br>PD_EXITMODE_NAK<br>PD_RESPONSE_NOT_SUPPORTED | Indicates the response type. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Timeout(micro second) to wait for receiving the Exit Mode command. |

### Result

None

### Examples

```
#Using default settings
$settings = PD_ExitMode_Settings
call PD_SetExitModeSetting( $settings )
```

## 5.1.127 PD_ExitMode

Starts *ExitMode* AMS.

### Format

```
Call PD_ExitMode(OrderedSetType, selectedSvid, modeIndex)
```

### Parameters

`OrderedSetType`

possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID

`modeIndex`

Indicates the mode index related to the specified SVID

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - `NAK` received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

**Examples**

```
call PD_ExitMode(PD_ORDERED_SET_TYPE_SOP, 0xFF00, 1)
```

### 5.1.128 PD_WaitForExitMode

Waits for user-defined timeout to receive EXITMODE command. It will respond to incoming messages as part of *ExitMode* AMS.

**Note:** The `PD_SetExitModeSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForExitMode()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - EXITMODE message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - `NAK` has been sent as response |

**Examples**

```
call PD_WaitForExitMode()
```

### 5.1.129 PD_Attention

Starts *Attention* AMS.

#### Format

```
Call PD_Attention( OrderedSetType, selectedSvid, modeIndex )
```

#### Parameters

`OrderedSetType`

Indicates the ordered set type. possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID

`modeIndex`

Indicates the mode index related to the specified SVID

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` are valid also (depends on the error type which has been occurred during sending data). |

#### Examples

```
call PD_Attention(PD_ORDERED_SET_TYPE_SOP, 0xFF01, 1 )
```

### 5.1.130 PD_AttentionVdo

Starts *Attention* AMS. It sends the specified VDO using he *Attention* command.

#### Format

```
Call PD_AttentionVdo( OrderedSetType, selectedSvid, modeIndex, PD_Generic_VDO $Vdo )
```

#### Parameters

`rderedSetType`

Indicates the ordered set type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

`selectedSvid`

Indicates the SVID

`modeId`

Indicates the mode index related to the specified SVID

`$Vdo`

Vendor defined data object. Should be from PD_VDO(Inherited from PD_Generic_VDO) type. (Refer to PD_VDO)

| Field Name | Description |
|---|---|
| Data | VDO data |

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket are valid also (depends on the error type which has been occurred during sending data). |

**Examples**

```
$vdo = PD_VDO
{
   Data = 0x00
}
call PD_AttentionVdo(PD_ORDERED_SET_TYPE_SOP, 0xFF01, 1, $vdo)
```

### 5.1.131 PD_SetDiscoveryProcessSetting

It has to be called prior calling the PD_PerformDiscoveryProcess function to take effect.

**Format**

```
Call PD_SetDiscoveryProcessSetting(PD_DiscoveryProcess_Settings $settings)
```

**Parameters**

$settings

Should be from PD_DiscoveryProcess_Settings type. Table below describes the PD_DiscoveryProcess_Settings template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| Discover_SOP_PP_During_SOP_P | PD_TRUE<br>PD_FALSE(Default) | Indicates whether perform SOP Double Prime discovery during SOP Prime discovery process or not. |
| SkipEnterMode | PD_FALSE(Default) | Indicates whether to skip the EnterMode phase or not. |

**Result**

None

**Examples**

```
#Using default settings
$settings = PD_DiscoveryProcess_Settings
call PD_SetDiscoveryProcessSetting( $settings )
```

### 5.1.132 PD_PerformDiscoveryProcess

Performs full discovery process.

**Note 1:** PD Exerciser supports only one(first) DiscoverSVIDs Ack message (up to 12 SVIDs).

**Note 2:** The PD_SetDiscoveryProcessSetting function may need to be called prior calling this function.

**Format**

```
Call PD_PerformDiscoveryProcess( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Indicates the ordered set type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
call PD_PerformDiscoveryProcess(PD_ORDERED_SET_TYPE_SOP)
```

### 5.1.133 PD_SetDisplayPortSetting

It has to be called prior calling the `PD_SetDisplayPortSetting_Cable`

The equivalent command of `PD_SetDisplayPortSetting` for an emulated cable.

**Format**

```
Call PD_SetDisplayPortSetting_Cable( PD_DisplayPort_Settings_SOPPrime $settings )
Call PD_SetDisplayPortSetting_Cable( PD_DisplayPort_Settings_SOPDPrime $settings )
```

PD_DisplayPort_UpdateStatus or `PD_DisplayPort_Configure` or `PD_WaitForDisplayPortStatus` or `Pd_WaitForDisplayPortStatus`_Cable

The equivalent command of `PD_WaitForDisplayPortStatus` for an emulated cable.

**Format**

```
Call PD_WaitForDisplayPortStatus_Cable( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

PD_WaitForDisplayPortConfigure or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetDisplayPortSetting( PD_DisplayPort_Settings $settings )
```

**Parameters**

`$settings`

Should be from `PD_DisplayPort_Settings` type. Table below describes the `PD_DisplayPort_Settings` template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| ConfigureResponse | PD_DISPLAYPORT_ACK(Default)<br>PD_DISPLAYPORT_NAK<br>PD_RESPONSE_NOT_SUPPORTED | Indicates the response for incoming Display Port Configure command. |

| | | |
|---|---|---|
| DisplayPortModeIndex | 0x01(Default) | Mode index related to the Display Port SVID. |
| StatusVdo | 0x00(Default) | Indicates the *Display Port Status Vendor Defined Data Object* which can be used in Display Port Update Status initiator or responder messages. |
| ConfigureVdo | 0x00(Default) | Indicates the *Display Port Configure Vendor Defined Data Object* which can be used in Display Port Configure initiator message. |
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(Default) | Timeout(micro second) to wait for receiving *Display Port Update Status or Configure* command. |

**Result**

None

**Examples**

```
#Using default settings
################################
$settings = PD_DisplayPort_Settings
call PD_SetDisplayPortSetting($settings)

#Set the StatusVdo
################################
$update_status = PD_VDM_DisplayPort_Status_VDO
{
    DFPD_UFPD_Connected     = PD_DISPLAYPORT_DFPD_CONNECTED
    PowerLow                = 0x00
    AdaptorEnabled          = 0x01
    MultiFunctionPreferred  = 0x01
    UsbConfigurationRequest = 0x00
    ExitDisplayModeRequest  = 0x00
    HPD_State               = 0x00
    IRQ_HPD                 = 0x00
    Reserved_DPS_1          = 0x00
}
$settings
{
    StatusVdo = $update_status
}
Call PD_SetDisplayPortSetting($settings)

#Set the ConfigureVdo to default
################################
$config = PD_VDM_DisplayPort_Configure_VDO
$settings
{
    ConfigureVdo = $config
}
Call PD_SetDisplayPortSetting($settings)
```

## 5.1.134 PD_SetDisplayPortSetting_Cable

The equivalent command of `PD_SetDisplayPortSetting` for an emulated cable.

**Format**

```
Call PD_SetDisplayPortSetting_Cable( PD_DisplayPort_Settings_SOPPrime $settings )
Call PD_SetDisplayPortSetting_Cable( PD_DisplayPort_Settings_SOPDPrime $settings )
```

## 5.1.135 PD_DisplayPort_UpdateStatus

Starts *DisplayPortUpdateStatus*(Structured VDM) AMS.

**Note:** The `PD_SetDisplayPortSetting` function may need to be called prior calling this function.

**Format**

```
Call PD_DisplayPort_UpdateStatus( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

### Examples

```
call PD_DisplayPort_UpdateStatus( PD_ORDERED_SET_TYPE_SOP_PRIME )
call PD_DisplayPort_UpdateStatus() # for SOP
```

## 5.1.136 PD_DisplayPort_Configure

Starts *DisplayPortConfigure*(Structured VDM) AMS.

**Note:** The `PD_SetDisplayPortSetting` function may need to be called prior calling this function.

### Format

```
Call PD_DisplayPort_Configure( OrderedSetType )
```

### Parameters

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - `NAK` received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

### Examples

```
call PD_DisplayPort_Configure( PD_ORDERED_SET_TYPE_SOP_PRIME )
call PD_DisplayPort_Configure() # for SOP
```

### 5.1.137 PD_WaitForDisplayPortStatus

Waits for user-defined timeout to receive DisplayPort STATUS command. It will respond to incoming messages as part of the *DisplayPortStatus*(Structured VDM) AMS.

**Note:** The `PD_SetDisplayPortSetting` function may need to be called prior calling this function.

#### Format

```
Call PD_WaitForDisplayPortStatus()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `UPDATE_STATUS` message not received |

#### Examples

```
call PD_WaitForDisplayPortStatus()
```

### 5.1.138 Pd_WaitForDisplayPortStatus_Cable

The equivalent command of `PD_WaitForDisplayPortStatus` for an emulated cable.

#### Format

```
Call PD_WaitForDisplayPortStatus_Cable( OrderedSetType )
```

#### Parameters

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### 5.1.139 PD_WaitForDisplayPortConfigure

Waits for user-defined timeout to receive DisplayPort CONFIGURE command. It will respond to incoming messages as part of *DisplayPortConfigure*(Structured VDM) AMS.

**Note:** The `PD_SetDisplayPortSetting` function may need to be called prior calling this function.

#### Format

```
Call PD_WaitForDisplayPortConfigure()
```

#### Parameters

None

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - Configure message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |

**Examples**

```
call PD_WaitForDisplayPortConfigure()
```

### 5.1.140 Pd_WaitForDisplayPortConfigure_Cable

The equivalent command of `PD_WaitForDisplayPortConfigure` for an emulated cable.

**Format**

```
Call PD_WaitForDisplayPortConfigure_Cable( OrderedSetType )
```

**Parameters**

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIM
```

### 5.1.141 PD_SetDiscoverIdentitySetting_Cable

It has to be called prior calling the `PD_WaitForDiscoverIdentity_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as a Cable Plug to be able to process this function.

**Format**

```
Call PD_SetDiscoverIdentitySetting_Cable( PD_DiscoverIdentity_Settings_SOPPrime $settings )
Call PD_SetDiscoverIdentitySetting_Cable( PD_DiscoverIdentity_Settings_SOPDPrime $settings )
```

**Parameters**

`$settings`

For SOP Prime, use `PD_DiscoverIdentity_Settings_SOPPrime` and for SOP Double Prime, use `PD_DiscoverIdentity_Settings_SOPDPrime`. Refer to `PD_SetDiscoverIdentitySetting` for more details. Only `DiscoverIdentityResponse` and `WaitTimeout` settings applied.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
PD_Set $PdGlobalSettings.EnableCableDPrimeEmulator = PD_TRUE
.
.
#Using default settings
$settings = PD_DiscoverIdentity_Settings_SOPPrime
call PD_SetDiscoverIdentitySetting_Cable( $settings )
```

```
$settings_dprime = PD_DiscoverIdentity_Settings_SOPDPrime
call PD_SetDiscoverIdentitySetting_Cable( $settings_dprime )
```

### 5.1.142 PD_WaitForDiscoverIdentity_Cable

Waits for user-defined timeout to receive DISCOVERIDENTITY command. It will respond to incoming messages as part of the *DiscoverIdentity* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The `Pd_WaitForDisplayPortConfigure_Cable`

The equivalent command of `PD_WaitForDisplayPortConfigure` for an emulated cable.

#### Format

```
Call PD_WaitForDisplayPortConfigure_Cable( OrderedSetType )
```

#### Parameters

`OrderedSetType`

Indicates the OrderedSet type. Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIM
```

PD_SetDiscoverIdentitySetting_Cable, `PD_AddDiscoverIdentityVDO_Cable` and `PD_ResetDiscoverIdentityVDO_Cable` functions may need to be called prior calling this function.

#### Format

```
Call PD_WaitForDiscoverIdentity_Cable( ordered_set )
```

#### Parameters

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVER_IDENTITY message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForDiscoverIdentity_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.143 PD_AddDiscoverIdentityVDO_Cable

Adds *DiscoverIdentity VDO*(for cable) to the PD Exerciser. It has to be called prior calling the PD_WaitForDiscoverIdentity_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

#### Format

```
Call PD_AddDiscoverIdentityVDO_Cable( PD_DiscoverIdentity_VDO $vdo, ordered_set )
```

#### Parameters

`$vdo`

Parameter type is PD_DiscoverIdentity_VDO. Refer to PD_VDM_Discover_Identity_Response for available DiscoverID VDOs.

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
#Add a Cable VDO
$vdo = PD_VDM_Discover_Identity_Cable_VDO
call PD_AddDiscoverIdentityVDO_Cable( $vdo, PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.144 PD_ResetDiscoverIdentityVDO_Cable

Clears *DiscoverIdentity VDOs*(for cable) in PD Exerciser. It should be called prior to call PD_AddDiscoverIdentityVDO_Cable function.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

#### Format

```
call PD_ResetDiscoverIdentityVDO_Cable( ordered_set )
```

#### Parameters

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
call PD_ResetDiscoverIdentityVDO_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.145 PD_SetDiscoverSVIDSetting_Cable

It has to be called prior calling the `PD_WaitForDiscoverSvids_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as a Cable Plug to be able to process this command.

#### Format

```
Call PD_SetDiscoverSVIDSetting_Cable( PD_DiscoverSvids_Settings_SOPPrime $settings )
Call PD_SetDiscoverSVIDSetting_Cable( PD_DiscoverSvids_Settings_SOPDPrime $settings )
```

#### Parameters

`$settings`

For SOP Prime, use `PD_DiscoverSvids_Settings_SOPPrime` and for SOP Double Prime, use `PD_DiscoverSvids_Settings_SOPDPrime`. Refer to `PD_SetDiscoverSVIDSetting` for more details. Only `DiscoverSvidsResponse` and `WaitTimeout` settings applied.

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
PD_Set $PdGlobalSettings.EnableCableDPrimeEmulator = PD_TRUE
.
.
#Using default settings
$settings = PD_DiscoverSvids_Settings_SOPPrime
call PD_SetDiscoverSVIDSetting_Cable( $settings )

$settings_dprime = PD_DiscoverSvids_Settings_SOPDPrime
call PD_SetDiscoverSVIDSetting_Cable( $settings_dprime )
```

### 5.1.146 PD_WaitForDiscoverSvids_Cable

Waits for user-defined timeout to receive DISCOVERSVID command. It will respond to incoming messages as part of *DiscoverSVIDs* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The `PD_SetDiscoverSVIDSetting_Cable`, `PD_AddSvid_Cable` and `PD_ResetSvids_Cable` functions may need to be called prior calling this function.

#### Format

```
Call PD_WaitForDiscoverSvids_Cable( ordered_set )
```

#### Parameters

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|

| | |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVER_SVIDS message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForDiscoverSvids_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.147 PD_AddSvid_Cable

Adds *SVIDs* (Cable Plug) to the PD Exerciser. It has to be called prior calling the PD_WaitForDiscoverSvids_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_AddSvid_Cable(value, ordered_set)
```

**Parameters**

`value`

SVID value to add

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
call PD_AddSvid_Cable(0xFF81, PD_ORDERED_SET_TYPE_SOP_PRIME)
```

### 5.1.148 PD_ResetSvids_Cable

Clears *SVIDs*(for cable) which is added to PD Exerciser. It should be called prior to call PD_AddSvid_Cable function.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_ResetSvids_Cable( ordered_set )
```

**Parameters**

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
```

```
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
call PD_ResetSvids_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.149 PD_SetDiscoverModeSetting_Cable

It has to be called prior calling the PD_WaitForDiscoverModes_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetDiscoverModeSetting_Cable( PD_DiscoverModes_Settings_SOPPrime $settings )
Call PD_SetDiscoverModeSetting_Cable( PD_DiscoverModes_Settings_SOPDPrime $settings )
```

**Parameters**

`$settings`

For SOP Prime, use PD_DiscoverModes_Settings_SOPPrime and for SOP Double Prime, use PD_DiscoverModes_Settings_SOPDPrime. Refer to PD_SetDiscoverModeSetting for more details. Only DiscoverModesResponse and WaitTimeout settings applied.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
PD_Set $PdGlobalSettings.EnableCableDPrimeEmulator = PD_TRUE
.
.
#Using default settings
$settings = PD_DiscoverModes_Settings_SOPPrime
call PD_SetDiscoverModeSetting_Cable( $settings )

$settings_dprime = PD_DiscoverModes_Settings_SOPDPrime
call PD_SetDiscoverModeSetting_Cable( $settings_dprime )
```

### 5.1.150 PD_WaitForDiscoverModes_Cable

Waits for user-defined timeout to receive DISCOVERMODE command. It will respond to incoming messages as part of *DiscoverModes* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The PD_SetDiscoverModeSetting_Cable, PD_AddModeVDO_Cable, PD_AddMode_Cable and PD_ResetModes_Cable functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForDiscoverModes_Cable( ordered_set )
```

### Parameters

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for **PD_SendPacket** and **PD_ReceivePacket** are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - DISCOVER_MODES message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |
| PD_SUBRESULT_RESPONSE_BUSY | Subresult - BUSY has been sent as response |

### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForDiscoverModes_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

## 5.1.151 PD_AddModeVDO_Cable

Adds *Mode*(for cable) with VDO to the PD Exerciser. It has to be called prior calling the **PD_WaitForDiscoverModes_Cable** or **PD_DelayAutoResponse** functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

### Format

```
Call PD_AddModeVDO_Cable(PD_Vdo $ModeVdo, ordered_set)
```

### Parameters

`$ModeVdo`

Should be from PD_Vdo type. Table below describes the PD_VDO template that can be use as ModeVdo (Refer to **PD_VDO**):

| Field Name | Description |
|---|---|
| Data | VDO |

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

### Result

None

### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
local $vdo_1 = PD_VDO
{
```

```
    Data = 0x01
}
call PD_AddModeVDO_Cable($vdo_1, PD_ORDERED_SET_TYPE_SOP_PRIME)
```

### 5.1.152 PD_AddMode_Cable

Adds *Mode*(Cable Plug) to the PD Exerciser. It has to be called prior calling the PD_WaitForDiscoverModes_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

#### Format

```
Call PD_AddMode_Cable(Mode, ordered_set)
```

#### Parameters

Mode

Mode to add

ordered_set

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
call PD_AddMode_Cable(0x00000001, PD_ORDERED_SET_TYPE_SOP_PRIME)
```

### 5.1.153 PD_ResetModes_Cable

Clears *Modes*(for cable) which are added to PD Exerciser. It should be called prior calling the PD_AddModeVDO_Cable and PD_AddMode_Cable functions.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

#### Format

```
Call PD_ResetModes_Cable( ordered_set )
```

#### Parameters

ordered_set

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
```

```
.
Call PD_ResetModes_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.154 PD_SetEnterModeSetting_Cable

It has to be called prior calling the PD_WaitForEnterMode_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

#### Format

```
Call PD_SetEnterModeSetting_Cable( PD_EnterMode_Settings_SOPPrime $settings )
Call PD_SetEnterModeSetting_Cable( PD_EnterMode_Settings_SOPDPrime $settings )
```

#### Parameters

`$settings`

For SOP Prime, use `PD_EnterMode_Settings_SOPPrime` and for SOP Double Prime, use `PD_EnterMode_Settings_SOPDPrime`. Refer to PD_SetEnterModeSetting for more details.

#### Result

None

#### Examples

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
PD_Set $PdGlobalSettings.EnableCableDPrimeEmulator = PD_TRUE
.
.
#Using default setting
$settings = PD_EnterMode_Settings_SOPPrime
call PD_SetEnterModeSetting_Cable( $settings )

$settings_dprime = PD_EnterMode_Settings_SOPDPrime
call PD_SetEnterModeSetting_Cable( $settings_dprime )
```

### 5.1.155 PD_WaitForEnterMode_Cable

Waits for user-defined timeout to receive ENTERMODE command. It will respond to incoming messages as part of *EnterMode* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The PD_SetEnterModeSetting_Cable function may need to be called prior calling this function.

#### Format

```
Call PD_WaitForEnterMode_Cable( ordered_set )
```

#### Parameters

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

#### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|

| | |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - ENTER_MODE message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForEnterMode_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.156 PD_SetExitModeSetting_Cable

It has to be called prior calling the PD_WaitForExitMode_Cable or PD_DelayAutoResponse functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetExitModeSetting_Cable( PD_ExitMode_Settings_SOPPrime $settings )
Call PD_SetExitModeSetting_Cable( PD_ExitMode_Settings_SOPDPrime $settings )
```

**Parameters**

`$settings`

For SOP Prime, use PD_ExitMode_Settings_SOPPrime and for SOP Double Prime, use PD_ExitMode_Settings_SOPDPrime. Refer to PD_SetExitModeSetting for more details.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
PD_Set $PdGlobalSettings.EnableCableDPrimeEmulator = PD_TRUE
.
.
#Using default settings
$settings = PD_ExitMode_Settings_SOPPrime
call PD_SetExitModeSetting_Cable( $settings )

$settings_dprime = PD_ExitMode_Settings_SOPDPrime
call PD_SetExitModeSetting_Cable( $settings_dprime )
```

### 5.1.157 PD_WaitForExitMode_Cable

Waits for user-defined timeout to receive EXITMODE command. It will respond to incoming messages as part of *ExitMode* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The PD_SetExitModeSetting_Cable function may need to be called prior calling this function.

**Format**

```
Call PD_WaitForExitMode_Cable( ordered_set )
```

**Parameters**

`ordered_set`

Possible values:

```
PD_ORDERED_SET_TYPE_SOP_PRIME
PD_ORDERED_SET_TYPE_SOP_DOUBLE_PRIME
```

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - EXIT_MODE message not received |
| PD_SUBRESULT_RESPONSE_NAK | Subresult - NAK has been sent as response |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForExitMode_Cable( PD_ORDERED_SET_TYPE_SOP_PRIME )
```

### 5.1.158 PD_SetManufacturerInfoDataBlock_Cable

Applicable to PD Rev 3.0 only. Sets *ManufacurerInfo Data Block*(for cable) to the PD Exerciser. It has to be called prior calling the `PD_WaitForGetManufacturerInfo_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetManufacturerInfoDataBlock_Cable( PD_ManufacturerInfoDataBlock
$manufacturer_info_db )
```

**Parameters**

`$manufacturer_info_db`

Parameter type is `PD_ManufacturerInfoDataBlock`. Refer to `PD_ManufacturerInfoMsg` for available fields of this type.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
$manufacturer_info_db = PD_ManufacturerInfoDataBlock
Call PD_SetManufacturerInfoDataBlock_Cable( $manufacturer_info_db )
```

### 5.1.159 PD_SetGetManufacturerInfoSetting_Cable

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForGetManufacturerInfo_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note :** PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetGetManufacturerInfoSetting_Cable( PD_GetManufacturerInfo_Settings $settings )
```

**Parameters**

`$settings`

Refer to `PD_SetGetManufacturerInfoSetting` for more details.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
$getmaninfo_setting = PD_GetManufacturerInfo_Settings
{
   WaitTimeout = 50000
}
Call PD_SetGetManufacturerInfoSetting_Cable( $getmaninfo_setting )
```

### 5.1.160 PD_WaitForGetManufacturerInfo_Cable

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive Manufacturer_Info message. It will respond to incoming messages as part of *GetManufacturerInfo* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The `PD_SetGetManufacturerInfoSetting_Cable` and `PD_SetManufacturerInfoDataBlock_Cable` functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForGetManufacturerInfo_Cable()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Manufacturer_Info` message not received. |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForGetManufacturerInfo_Cable()
```

### 5.1.161 PD_SetSecurityResponseDataBlock_Cable

Applicable to PD Rev 3.0 only. Sets the *SecurityResponse Data Block*(for cable) to the PD Exerciser. It has to be called prior calling the `PD_WaitForSecurityRequest_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetSecurityResponseDataBlock_Cable( PD_SecurityResponseDB $security_resp_db )
```

**Parameters**

`$security_resp_db`

Parameter type is `PD_SecurityResponseDB`. Refer to `PD_SecurityResponseMsg` for available types which are derived from this type.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
$security_resp_db = PD_SRPDB_Certificate
Call PD_SetSecurityResponseDataBlock_Cable( $security_resp_db )
```

### 5.1.162 PD_SetSecurityRequestSetting_Cable

Applicable to PD Rev 3.0 only. It has to be called prior calling the `PD_WaitForSecurityRequest_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetSecurityRequestSetting_Cable( PD_SecurityRequest_Settings $settings )
```

**Parameters**

`$settings`

Refer to `PD_SetSecurityRequestSetting` for more details.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
$secreq_settings = PD_SecurityRequest_Settings
{
   WaitTimeout = 50000
}
Call PD_SetSecurityRequestSetting( $secreq_settings )
```

### 5.1.163 PD_WaitForSecurityRequest_Cable

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive Security_Request message. It will respond to incoming messages as part of *SecurityRequest* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The `PD_SetSecurityResponseDataBlock_Cable` and `PD_SetSecurityRequestSetting_Cable` functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForSecurityRequest_Cable()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - Security_Request message not received. |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForSecurityRequest_Cable()
```

### 5.1.164 Pd_SetGetStatusSetting_Cable

Applicable to PD Rev 3.0 only. It has to be called prior calling the `Pd_WaitForGetStatus_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call Pd_SetGetStatusSetting_Cable( Pd_GetStatus_Settings $settings )
```

**Parameters**

`$settings`

Parameter type is `PD_GetStatus_Settings`. Refer to `PD_SetGetStatusSetting` for available settings:

**Result**

None

**Examples**

```
$getstatus_setting = PD_GetStatus_Settings
{
   WaitTimeout = 250000
}
Call Pd_SetGetStatusSetting_Cable( $getstatus_setting )
```

### 5.1.165 PD_SetStatusDataBlock_Cable

Applicable to PD Rev 3.0 only. Sets the *Cable Status Data Block* in PD Exerciser. It has to be called prior calling the `Pd_WaitForGetStatus_Cable` or `PD_DelayAutoResponse` functions to take effect.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetStatusDataBlock_Cable( Pd_StatusDataBlock_Cable $status_db )
```

**Parameters**

`$status_db`

Parameter type is `Pd_StatusDataBlock_Cable`. Refer to `PD_StatusMsg_Cable` for available fields.

**Result**

None

**Examples**

```
$cable_status_db = Pd_StatusDataBlock_Cable
Call PD_SetStatusDataBlock_Cable( $cable_status_db )
```

### 5.1.166 Pd_ResetStatusDataBlock_Cable

Applicable to PD Rev 3.0 only. Clears the *Cable Status Data Block* in PD Exerciser. It should be called prior calling the `PD_SetStatusDataBlock_Cable` function.

**Note** - PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call Pd_ResetStatusDataBlock_Cable()
```

**Parameters**

None

**Result**

None

**Examples**

```
Call Pd_ResetStatusDataBlock_Cable()
```

### 5.1.167 Pd_WaitForGetStatus_Cable

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive `Get_Status` message. It will respond to incoming messages as part of *GetStatus* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The `Pd_SetGetStatusSetting_Cable`, `PD_SetStatusDataBlock_Cable` and `Pd_ResetStatusDataBlock_Cable` functions may need to be called prior calling this function.

**Format**

```
Call Pd_WaitForGetStatus_Cable( )
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for PD_SendPacket and PD_ReceivePacket are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - Get_Status message not received. |

**Examples**

```
Call Pd_WaitForGetStatus_Cable()
```

### 5.1.168 PD_SetEnterUSBSetting_Cable

Applicable to PD Rev 3.0 only. It has to be called prior calling the PD_WaitForEnterUSB_Cable or PD_DelayAutoResponse functions to take effect.

**Note :** PD Exerciser should also act as Cable Plug to be able to process this command.

**Format**

```
Call PD_SetEnterUSBSetting_Cable( PD_EnterUSB_Settings $settings )
```

**Parameters**

`$settings`

Refer to PD_SetEnterUSBSetting for more details.

**Result**

None

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
$enter_usb_setting = PD_EnterUSB_Settings
{
   WaitTimeout = 50000
}
Call PD_SetEnterUSBSetting_Cable( $enter_usb_setting )
```

### 5.1.169 PD_WaitForEnterUSB_Cable

Applicable to PD Rev 3.0 only. Waits for user-defined timeout to receive EnterUSB message. It will respond to incoming messages as part of *EnterUSB* AMS.

**Note 1:** PD Exerciser should also act as Cable Plug to be able to process this command.

**Note 2:** The PD_SetEnterUSBSetting_Cable functions may need to be called prior calling this function.

**Format**

```
Call PD_WaitForEnterUSB_Cable()
```

**Parameters**

None

**Result**

User can evaluate the command results(including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult – `EnterUSB` message not received. |

**Examples**

```
PD_Set $PdGlobalSettings.EnableCableEmulator = PD_TRUE
.
.
Call PD_WaitForEnterUSB_Cable()
```

### 5.1.170 Pd_SetHardResetSetting

It has to be called prior calling the `PD_DelayAutoResponse` function or any other *HighLevel Transaction Engine™* functions to take effect.

**Format**

```
Call Pd_SetHardResetSetting( Pd_HardResetSettings $settings )
```

**Parameters**

`$settings`

Should be from `Pd_HardResetSettings` type. Following are the available fields of this packet template:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeOut | `PD_DEFAULT_TIMEOUT_INFINIT`(Default) | Indicates the timeout for waiting to receive the `HardReset`. Only applies to the `Pd_WaitForHardReset` function. |
| NoResponse | `PD_FALSE`(Default) | Indicates whether the PD Exerciser should respond to received `HardReset`s or not. It applies to all *HighLevel Transaction Engine™* functions as well as `PD_DelayAutoResponse` function. |

**Result**

None

**Examples**

```
#Ignore all received HardResets
$settings = Pd_HardResetSettings
{
   NoResponse = PD_TRUE
}
call Pd_SetHardResetSetting( $settings )
```

### 5.1.171 Pd_WaitForHardReset

Waits for user-defined timeout to receive `HardaReset` signal. It handles the `HardReset` signal.

**Note:** The `Pd_SetHardResetSetting` function may need to be called prior calling this function.

**Format**

```
Call Pd_WaitForHardReset()
```

**Parameters**

None

### Result

User can evaluate the command results(including sub-results) using `IfMatched/ElseMatched` command.

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_ReceivePacket` are valid also (depends on the error type which has been occurred while receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `HardReset` signal not received |

### Examples

```
call Pd_WaitForHardReset()
```

## 5.1.172 PD_AddEPRSourceCap

It adds an EPR Source Capability to the PD Exerciser to be used by `PD_NegotiatePower_Source` or `PD_NegotiatePower` functions. Before adding a group of EPR source caps make sure that there is no unwanted EPR source cap in the list by calling `PD_ResetEPRSinkCaps`.

### Format

```
Call PD_AddEPRSourceCap(PD_PowerDataObject $PowerDataObject)
```

### Parameters

`$PowerDataObject`

Refer to `PD_EPRSourceCapabilitiesMessage` for available EPR source power data objects.

### Result

None

### Examples

```
local $power_data_object = PD_PowerDataObjectFixedSupply_Source
{
   MaxCurrent_10mAUnits = 300
   Voltage_50mVUnits = 800
}
call PD_AddEPRSourceCap($power_data_object)
```

## 5.1.173 PD_ResetEPRSourceCaps

Clears all EPR Source Capabilities added to PD Exerciser.

### Format

```
Call PD_ResetEPRSourceCaps()
```

### Parameters

None

### Result

None

### Examples

```
call PD_ResetEPRSourceCaps()
```

### 5.1.174 PD_AddEPRSinkCap

It adds an EPR Sink Capabilities to PD Exerciser to be used by PD_NegotiatePower_Sink , PD_NegotiatePower or PD_WaitForNegotiatePower. Before adding a group of EPR sink caps make sure that there is no unwanted EPR sink cap in the list by calling PD_ResetEPRSinkCaps function.

Note - By default there is one pre-defined sink cap in the list.

**Format**

```
Call PD_AddEPRSinkCap(PD_PowerDataObject $PowerDataObject)
```

**Parameters**

`$PowerDataObject`

Refer to PD_EPRSinkCapabilitiesMessage for available sink power data objects.

**Result**

None

**Examples**

```
local $power_data_object = PD_PowerDataObjectFixedSupply_Sink
{
   OperationalCurrent_10mAUnits = 500
   Voltage_50mVUnits = 1000
}
call PD_AddEPRSinkCap($power_data_object)
```

### 5.1.175 PD_ResetEPRSinkCaps

Clears all EPR Sink Capabilities defined in PD Exercise.

**Format**

```
Call PD_ResetEPRSinkCaps()
```

**Parameters**

None

**Result**

None

**Examples**

```
call PD_ResetEPRSinkCaps()
```

### 5.1.176 PD_ SetGetEPRSourceCapSetting

It has to be called prior to call `PD_WaitForGetEPRSourceCapabilities` or `PD_DelayAutoResponse` functions to take effect.

#### Format

```
Call PD_SetGetEPRSourceCapSetting( PD_GetEPRSourceCapability_Settings $settings )
```

#### Parameters

`$settings`

Setting type is `PD_GetEPRSourceCapability_Settings`. The available field of this type is:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait timeout (micro second) for receiving GetEPRSourceCap message. |

#### Result

None

#### Examples

```
$geteprsrccap_setting = PD_GetEPRSourceCapability_Settings
{
  WaitTimeout = 50000
}
Call PD_SetGetEPRSourceCapSetting( $geteprsrccap_setting )
```

### 5.1.177 PD_GetEPRSourceCapabilities

Starts GetEPRSourceCapabilities AMS.

#### Format

```
Call PD_GetEPRSourceCapabilities()
```

#### Parameters

None

#### Result

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No message received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

#### Examples

```
Call PD_GetEPRSourceCapabilities()
```

### 5.1.178 PD_WaitForGetEPRSourceCapabilities

It waits for the DUT to start EPR*GetSourceCap* AMS.

**Note:** The `PD_SetGetEPRSourceCapSetting` function can be called prior calling this function to set a timeout.

#### Format

```
Call PD_WaitForGetEPRSourceCapabilities()
```

#### Parameters

None

#### Result

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_EPR_Source_Cap` message not received |
| PD_SUBRESULT_REQUEST_MSG_INVALID_INDEX | Subresult - Invalid index in request message |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message has been sent as response |

#### Examples

```
Call PD_WaitForGetEPRSourceCapabilities()
```

### 5.1.179 PD_SetGetEPRSinkCapSetting

It has to be called prior to call `PD_WaitForGetEPRSinkCapabilities` or `PD_DelayAutoResponse` functions to take effect.

#### Format

```
Call PD_SetGetEPRSinkCapSetting( PD_GetEPRSinkCapability_Settings $settings )
```

#### Parameters

`$settings`

Setting type is `PD_GetEPRSinkCapability_Settings`. The available field of this type is:

| Field Name | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait timeout (micro second) for receiving GetEPRSinkCap message. |

#### Result

None

#### Examples

```
$geteprsnkcap_setting = PD_GetEPRSinkCapability_Settings
```

```
{
   WaitTimeout = 50000
}
Call PD_SetGetEPRSinkCapSetting( $geteprsnkcap_setting )
```

### 5.1.180 PD_GetEPRSinkCapabilities

Starts GetEPRSinkCapabilities AMS.

**Format**

```
Call PD_GetEPRSinkCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No message received as response |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |

**Examples**

```
Call PD_GetEPRSinkCapabilities()
```

### 5.1.181 PD_WaitForGetEPRSinkCapabilities

It waits for the DUT to start *GetEPRSinkCapabilities* AMS.

**Note:** The `PD_SetGetEPRSinkCapSetting` function can be called prior calling this function t set a timeout.

**Format**

```
Call PD_WaitForGetEPRSinkCapabilities()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_MSG_NOT_RECEIVED | Subresult - `Get_Sink_Cap` message not received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message has been sent as response |

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

### Examples

```
Call PD_WaitForGetEPRSinkCapabilities()
```

## 5.1.182 PD_SetEnterEPRModeSetting

It has to be called prior to call PD_WaitForEnterEPRMode or PD_DelayAutoResponse functions to take effect.

### Format

```
Call PD_SetEnterEPRModeSetting( PD_EnterEPRMode_Settings $settings )
```

### Parameters

$settings

Setting type is PD_EnterEPRMode_Settings. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | PD_DEFAULT_TIMEOUT_INFINIT(default) | Wait Timeout (micro second) to receive EPR_mode with Enter action message. |
| EnterPDP | 0(default) | EPR Sink Operational PDP |
| AckFailReason | PD_EPR_ENTER_FAIL_DATA_UDEFINED(default) | If set, enter EPR mode fails with the specified reason on the acknowledge phase |
| EnterFailReason | PD_EPR_ENTER_FAIL_DATA_UDEFINED(default) | If set, enter EPR mode fails with the specified reason on the last phase |
| SkipPowerNegotiation | PD_TRUE,<br>PD_FALSE (default) | To skip power negotiation after entering EPR mode. |
| SkipCableDiscovery | PD_TRUE,<br>PD_FALSE (default) | To skip cable discovery process during enterin EPR mode. |
| SkipEnterSucceeded | PD_TRUE,<br>PD_FALSE (default) | To skip sending “Enter Succeeded” for error injection. |

### Result

None

### Examples

```
$entereprmode_setting = PD_EnterEPRMode_Settings
{
   WaitTimeout = 50000
}
Call PD_SetEnterEPRModeSetting( $entereprmode_setting )
```

## 5.1.183 PD_EnterEPRMode

Starts *EnterEPRMode* AMS.

### Format

```
Call PD_EnterEPRMode()
```

### Parameters

None

### Result

User can evaluate the command results (including sub-results) using IfMatched/ElseMatched command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_EnterEPRMode()
```

### 5.1.184 PD_WaitForEnterEPRMode

It waits for the DUT to start *EnterEPRMode* AMS.

Note: `The PD_SetEnterEPRModeSetting` function can be called prior calling this function to set a timeout.

**Format**

```
Call PD_WaitForEnterEPRMode()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_WaitForEnterEPRMode()
```

### 5.1.185 PD_SetExitEPRModeSetting

It has to be called prior to call `PD_WaitForExitEPRMode` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetExitEPRModeSetting( PD_ExitEPRMode_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `PD_ExitEPRMode_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait Timeout (micro second) to receive `EPR_mode` message with Exit action. |

| | | |
|---|---|---|
| DelayBeforeExit | 0 | Defines the delay before exiting EPR mode |
| SkipNegotiationBeforeExit | PD_TRUE,<br>PD_FALSE (default) | To skip SPR power negotiation before exiting EPR mode |
| SkipNegotiationAfterExit | PD_TRUE,<br>PD_FALSE (default) | To skip power negotiation after exiting EPR mode. |

**Result**

None

**Examples**

```
local $exiteprmode_setting = PD_ExitEPRMode_Settings
{
   WaitTimeout = 50000
}
Call PD_SetExitEPRModeSetting( $exiteprmode_setting )
```

### 5.1.186 PD_ExitEPRMode

Starts *ExitEPRMode* AMS.

**Format**

```
Call PD_ExitEPRMode()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_ExitEPRMode()
```

### 5.1.187 PD_WaitForExitEPRMode

It waits for the DUT to start *ExitEPRMode* AMS.

Note: The `PD_SetExitEPRModeSetting` function can be called prior calling this function to set a timeout.

**Format**

```
Call PD_WaitForExitEPRMode()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_WaitForExitEPRMode()
```

## 5.1.188 PD_SetEPRKeepAliveSetting

It has to be called prior to call `PD_WaitForEPRKeepAlive` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetEPRKeepAliveSetting( PD_EPRKeepAlive_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `PD_EPRKeepAlive_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait Timeout (micro second) to receive `EnterUSB` message. |
| DisableAutoKeepAlive | `PD_FALSE`(default)<br>PD_TRUE | Disable Sending `EPRKeepAlive` message automatically during `DelayAutoResponse` |

**Result**

None

**Examples**

```
local $epr_setting = PD_EPRKeepAlive_Settings
{
   WaitTimeout = 50000
}
Call PD_SetEPRKeepAliveSetting( $EPRKeepAlive_setting )
```

## 5.1.189 PD_EPRKeepAlive

Starts *EPRKeepAlive* AMS.

**Format**

```
Call PD_EPRKeepAlive()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_EPRKeepAlive()
```

## 5.1.190 PD_WaitForEPRKeepAlive

It waits for the DUT to start *EPRKeepAlive* AMS.

Note: The `PD_SetEPRKeepAliveSetting` function can be called prior calling this function to set a timeout.

**Format**

```
Call PD_WaitForEPRKeepAlive()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_WaitForEPRKeepAlive()
```

## 5.1.191 PD_SetGetRevisionSetting

It has to be called prior to call `PD_WaitForGetRevision` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetGetRevisionSetting( PD_GetRevision_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `PD_GetRevision_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait Timeout (micro second) to receive `EnterUSB` message. |

**Result**

None

**Examples**

```
$GetRevision_setting = PD_GetRevision_Settings
{
   WaitTimeout = 50000
}
Call PD_SetGetRevisionSetting( $GetRevision_setting )
```

### 5.1.192 PD_GetRevision

Starts *GetRevision* AMS.

**Format**

```
Call PD_GetRevision()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_GetRevision()
```

### 5.1.193 PD_WaitForGetRevision

It waits the DUT to start *GetRevision* AMS.

Note: The `PD_SetGetRevisioSetting` function can be called prior calling this function to set a timeout.

**Format**

```
Call PD_WaitForGetRevision()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_WaitForGetRevision()
```

## 5.1.194 PD_SetGetSourceInfoSetting

It has to be called prior to call `PD_WaitForGetSourceInfo` or `PD_DelayAutoResponse` functions to take effect.

**Format**

```
Call PD_SetGetSourceInfoSetting( PD_GetSourceInfo_Settings $settings )
```

**Parameters**

`$settings`

Setting type is `PD_GetSourceInfo_Settings`. Available fields for this type are:

| Field Names | Possible/Default Values | Description |
|---|---|---|
| WaitTimeout | `PD_DEFAULT_TIMEOUT_INFINIT`(default) | Wait Timeout (micro second) to receive `EnterUSB` message. |

**Result**

None

**Examples**

```
$GetSourceInfo_setting = PD_GetSourceInfo_Settings
{
   WaitTimeout = 50000
}
Call PD_SetGetSourceInfoSetting( $GetSourceInfo_setting )
```

## 5.1.195 PD_GetSourceInfo

Starts *GetSourceInfo* AMS.

**Format**

```
Call PD_GetSourceInfo()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |
| PD_SUBRESULT_RESPONSE_NOT_SUPPORTED | Subresult - `Not_Supported` message received as response |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` received as the response **(in Revision 2.0)** |

**Examples**

```
Call PD_GetSourceInfo()
```

### 5.1.196 PD_WaitForGetSourceInfo

It waits for the DUT to start *GetSourceInfo* AMS.

Note: The `PD_SetGetSourceInfoSetting` function can be called prior calling this function to set a timeout.

**Format**

```
Call PD_WaitForGetSourceInfo()
```

**Parameters**

None

**Result**

User can evaluate the command results (including sub-results) using `IfMatched/ElseMatched` command.

List of result values:

| Result Value | Description |
|---|---|
| PD_RESULT_OK | Command succeeded |
| PD_RESULT_FAILED | Command failed. In this case corresponding sub results for `PD_SendPacket` and `PD_ReceivePacket` are valid also (depends on the error type which has been occurred during sending or receiving data). |
| PD_SUBRESULT_RESPONSE_REJECT | Subresult - `Reject` message has been sent as response |
| PD_SUBRESULT_RESPONSE_TIMEOUT | Subresult - No response received. |

**Examples**

```
Call PD_WaitForGetSourceInfo()
```

## 5.2 Auto Responses Capability

The following commands will respond to all incoming Power Delivery messages automatically according to current operational settings. In addition to tjhese commands, at the start of each High-Level command Auto-Response is activated.

### 5.2.1 PD_DelayAutoResponse

**Format**

```
Call PD_DelayAutoResponse( duration_micro_Sec )
```

**Parameters**

`duration_micro_Sec`

Command waits for maximum specified duration and responses to received packet automatically.

**Examples**

```
call PD_DelayAutoResponse( 1000 )
```

### 5.2.2 PD_PauseAutoResponse

Responsive PD Exerciser pause which waits until `ResumePDGeneration()` called by the *Automation Application*. By calling this function, an *Automation Application* get notified that PD Exerciser Engine is paused. Refer to ***USBAnalyzerAutomationManual*** for more information on `WaitForPDGenerationPauseTag()` and `ResumePDGeneration()` API calls.

**Note:** All the rules which are applied to the `PD_DelayAutoResponse` function, will apply to this function too.

**Format**

```
Call PD_PauseAutoResponse( tag )
```

**Parameters**

`tag`

Makes each call to `PD_PauseAutoResponse` unique.

**Examples**

```
# PD Exerciser Script example:
call PD_PauseAutoResponse( 1 )

' WSH example:
Set Analyzer = WScript.CreateObject("CATC.USBTracer")

Analyzer.WaitForPDGenerationPauseTag 15000, diff_time, pauseTag, isGenDone

If pauseTag <> 0 Then
   WScript.Quit
End If

Analyzer.ResumePDGeneration
```

**WARNING:** EAR99 Technology Subject to Restrictions Contained on the Cover Page.

# 6 Appendix A:

## How to Contact Teledyne LeCroy

| | |
|---|---|
| Send e-mail… | psgsupport@teledyne.com |
| Contact support… | teledynelecroy.com/support/contact |
| Visit Teledyne LeCroy's web site… | teledynelecroy.com |
| Tell Teledyne LeCroy… | Report a problem to Teledyne LeCroy Support via e-mail by selecting **Help > Tell Teledyne LeCroy** from the application toolbar. This requires that an e-mail client be installed and configured on the host machine. |