Verification Script Engine
for
Teledyne LeCroy USB Protocol Suite™
Reference Manual

Manual Version 9.08

For USB Protocol Suite v10.10 and above

June 2025

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

Teledyne, Teledyne LeCroy, LeCroy, CATC, and *USB Protocol Suite* are trademarks of Teledyne LeCroy.

Microsoft and *Windows* are registered trademarks of Microsoft Inc.

All other trademarks are property of their respective companies.

Copyright

Contents

1 Introduction

This document describes the Teledyne LeCroy USB Protocol Suite™ Verification Script Engine (VSE), which allows users to perform complicated custom analyses of traffic recorded using the Teledyne LeCroy protocol analyzers.

The VSE allows you to create verfication scripts that ask the application to send specific "events" (see Chapter 7: "Verification Script Engine Events") that occur in the recorded trace to a function that can process them. This function, along with other logic in a verification script, can be used to evaluate the sequence of events (timing, data, or both) in accordance with user-defined conditions and perform post-processing tasks; such as exporting key information to external files (in text or binary format) or sending special Automation™/COM notifications to client applications.

Verfication scripts are based on the CATC Scripting Language (CSL). It is recommended to review the "CATC Scripting Language Reference Manual for CATC USB Analzyers" (USBScriptDecodeManual.pdf). This document covers the semantics, functions, and other specifics for the VSE, however the sematics of the scripting language itself, such as data types and other helper functions and primitives can be found in the CSL reference manual.

The VSE fully utilizes the high performance and intelligence of Teledyne LeCroy decoding capabilities, making processing of information easy and fast. VSE can:

- Retrieve information about any field in frame headers, contents of frame payloads, serial data, and bus states.
- Make complex timing calculations between different events in pre-recorded traces.
- Filter-in or filter-out data with dynamically changing filtering conditions.
- Port information to a special output window.
- Save data to text or binary files.
- Send data to COM clients connected to the application.

2 Verification Script Structure

Writing verification scripts is easy, if you understand how the application interacts with running scripts and if you follow some rules imposed by the verification script syntax.

The main file with the text of the verification script must have extension **.vse**. It must be located in the subfolder **..\Scripts\VFScripts** of the main folder. Some other files must be included in the main script file using the directive **%include**.

The following schema presents a common structure of a verification script. It is similar to the script template **VSTemplate.vs_**, which is included with VSE.

```
#
# VS1.vse
#
# Verification script
#
# Brief Description:
# Verify something.
#
#############################################################################################
#                                          Module info                                      #
#############################################################################################
#       Filling of this block is necessary for proper verification script operation...     #
```

```
set DecoderDesc = "<Your Verification Script description>"; # Optional
```

```
#
# Include main Verification Script Engine definitions.
#
%include  "VSTools.inc"                 # Should be set for all verification scripts.
```

```
#################################################################################
#                              Global Variables and Constants                   #
#################################################################################

# Define your verification script-specific global variables and constant in this section.
# (Optional)

const MY_GLOBAL_CONSTANT = 10;
set g_MyGlobalVariable   = 0;

#   OnStartScript()                                                             #
#################################################################################
#                                                                               #
# Main intialization routine for setting up all necessary                       #
# script parameters before running the script.                                  #
#                                                                               #
#################################################################################
OnStartScript()
{
  ###############################################################################
  # Specify in the body of this function the initial values for global variables #
  # and what kinds of trace events should be passed to the script.              #
  # (By default, only Primitive events from all channels                        #
  # are passed to the script.                                                   #
  #                                                                             #
  # For details about how to specify the kind of events to pass to the script,  #
  # please see the topic 'Sending Functions'.                                   #
  #                                                                             #
  # OPTIONAL.                                                                  #
  ###############################################################################

  # Uncomment the line below if you want to disable output from
  # ReportText()-functions.
  #
  # DisableOutput();
}
```

```
#################################################################################################
#   ProcessEvent()                                                                              #
#===============================================================================================#
#                                                                                               #
#                                                                                               #
#===============================================================================================#
# Main script function called by the application when the next waited event                     #
# occurs in the evaluated trace.                                                                #
#                                                                                               #
# !!! REQUIRED !!! MUST BE IMPLEMENTED IN THE VERIFICATION SCRIPT                               #
#===============================================================================================#
#                                                                                               #
ProcessEvent()
{

  # Write the body of this function depending on your needs.

  Return Complete();
}

#   OnFinishScript()                                                                            #
#                                                                                               #
#===============================================================================================#
# Main script function called by the application when the script has completed                  #
# running. Specify in this function some resetting procedures for a successive run              #
# of this script.                                                                               #
#                                                                                               #
#  OPTIONAL.                                                                                    #
#===============================================================================================#
OnFinishScript()
{
  return 0;
}

#   Additional script functions.                                                                #
#===============================================================================================#
#                                                                                               #
# Write your own script-specific functions here.                                                #
#                                                                                               #
#===============================================================================================#
MyFunction(arg)
{
  if(arg == "Blah") return 1;
  return 0;
}
```

3 Interaction between Application and Verification Script

The following steps describe the interaction between the application and a verification script run over a recorded trace:

1. Before sending any information to the script main processing function **ProcessEvent()** (which must be present in any verification script),VSE looks for function **OnStartScript()** and calls it if it is found. In this function, some setup routines can be made, like specifying the channels, specifying the trace events to pass to the script, and setting up initial values of the global script-specific global variables.

2. Then VSE goes through the recorded trace and checks if the current frame in the trace meets specified sending criteria. If it does, VSE calls the script main processing function **ProcessEvent(),** providing some information about the current event in the script input context variables.

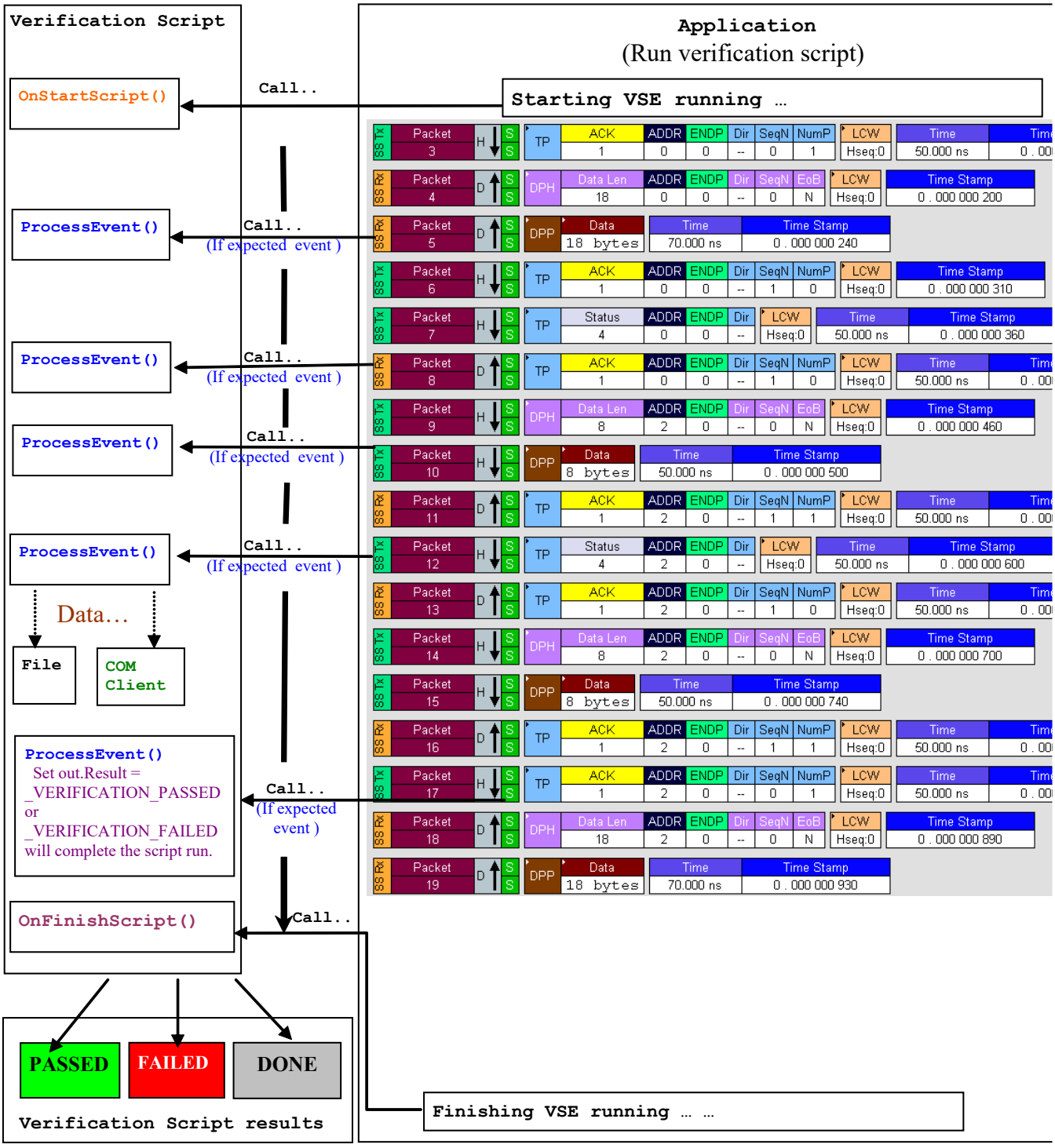
   (See the topic "Input context variables" below in this document for a full description of verification script input context variables.)

3. **ProcessEvent()** is the main verification routine, in which all processing of incoming trace events is done. Basically, when the whole verification program consists of a few stages, this function processes the event sent to the script, verifies that information contained in the event is appropriate for the current stage, and decides if VSE should continue script running. If the whole result is clear on the current stage, it tells VSE to complete execution of the script.
   The completion of the test before the whole trace has been evaluated is usually done by setting the output context variable:
   out.Result = _VERIFICATION_PASSED or **_VERIFICATION_FAILED**.
   (See the topic "Output context variables" below in this document for a full description of verification script output context variables.)

   Note: Not only does a verification script verify recorded traces by some criteria, but it is also possible to extract some information of interest and post-process it later by third-party applications. (There is a set of script functions to save extracted data in text or binary files or send it to other applications via COM/Automation™ interfaces.)

1 When script running is finished, VSE looks for the function **OnFinishScript()** and calls it if it is found. In this function, some resetting procedures can be done.

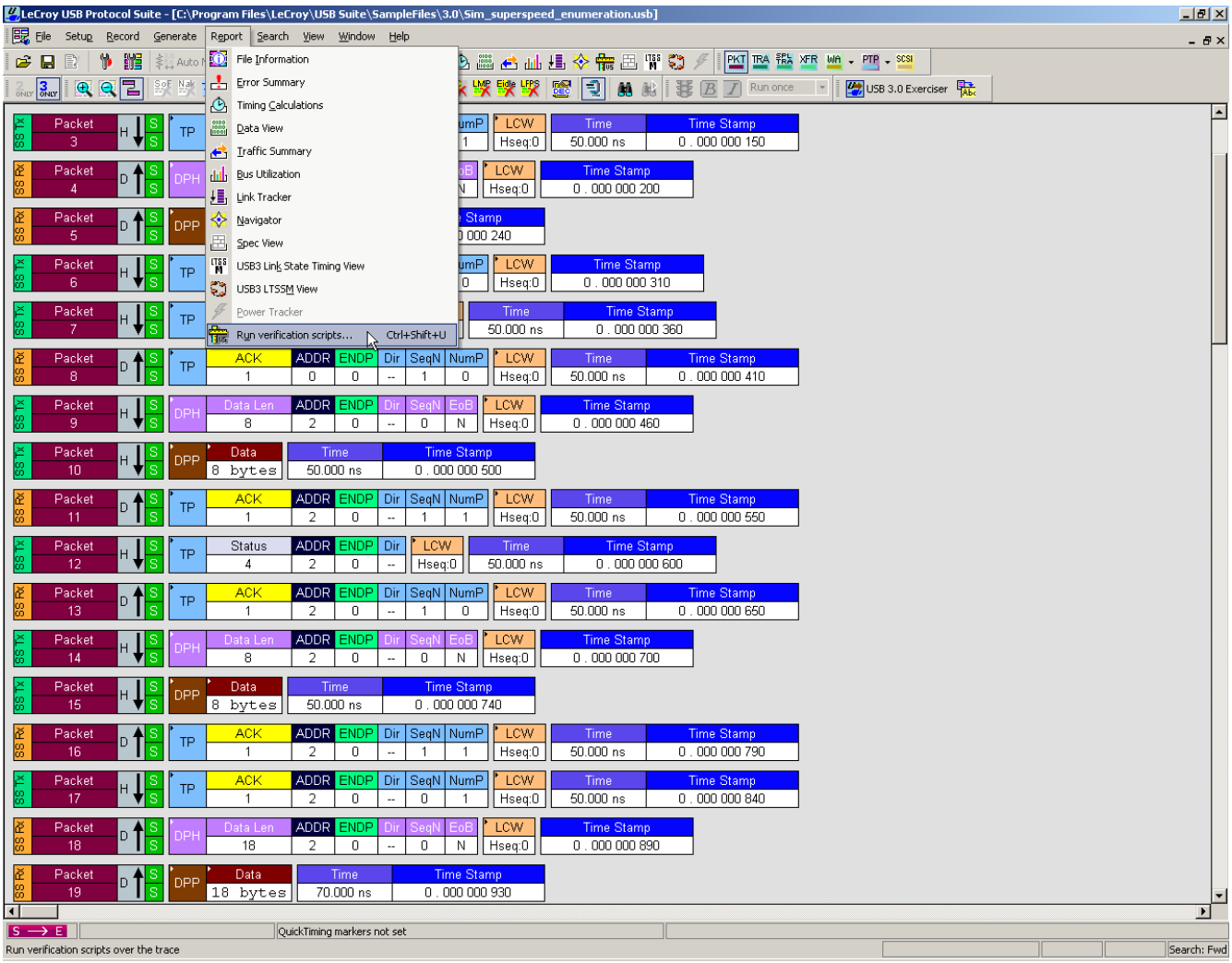
The following picture presents the interaction between the application and a running verification script:



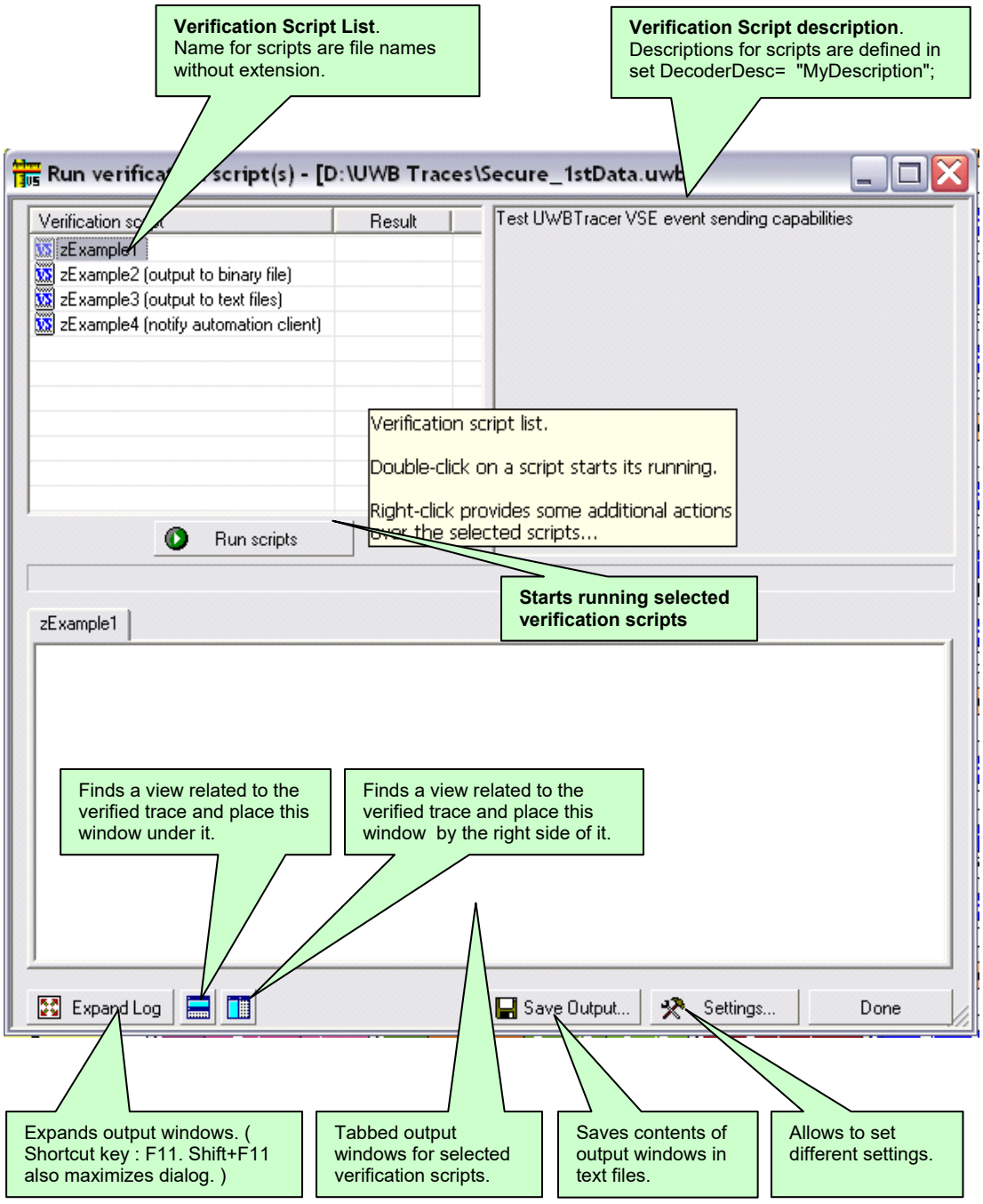
Note: Verification script result **<DONE>** means that the script is intended for extracting and displaying some information from recorded traces only, and you do not care about results. To specify that your script is for displaying information only, call the function **ScriptForDisplayOnly()** somewhere in your script (in **OnStartScript()**, for instance).

4 Running Verification Scripts from Application

To run a verification script over a trace, you select the main menu item **Report > Run verification scripts** or click the **Run verification scripts** button on the main tool bar (if it is not hidden):

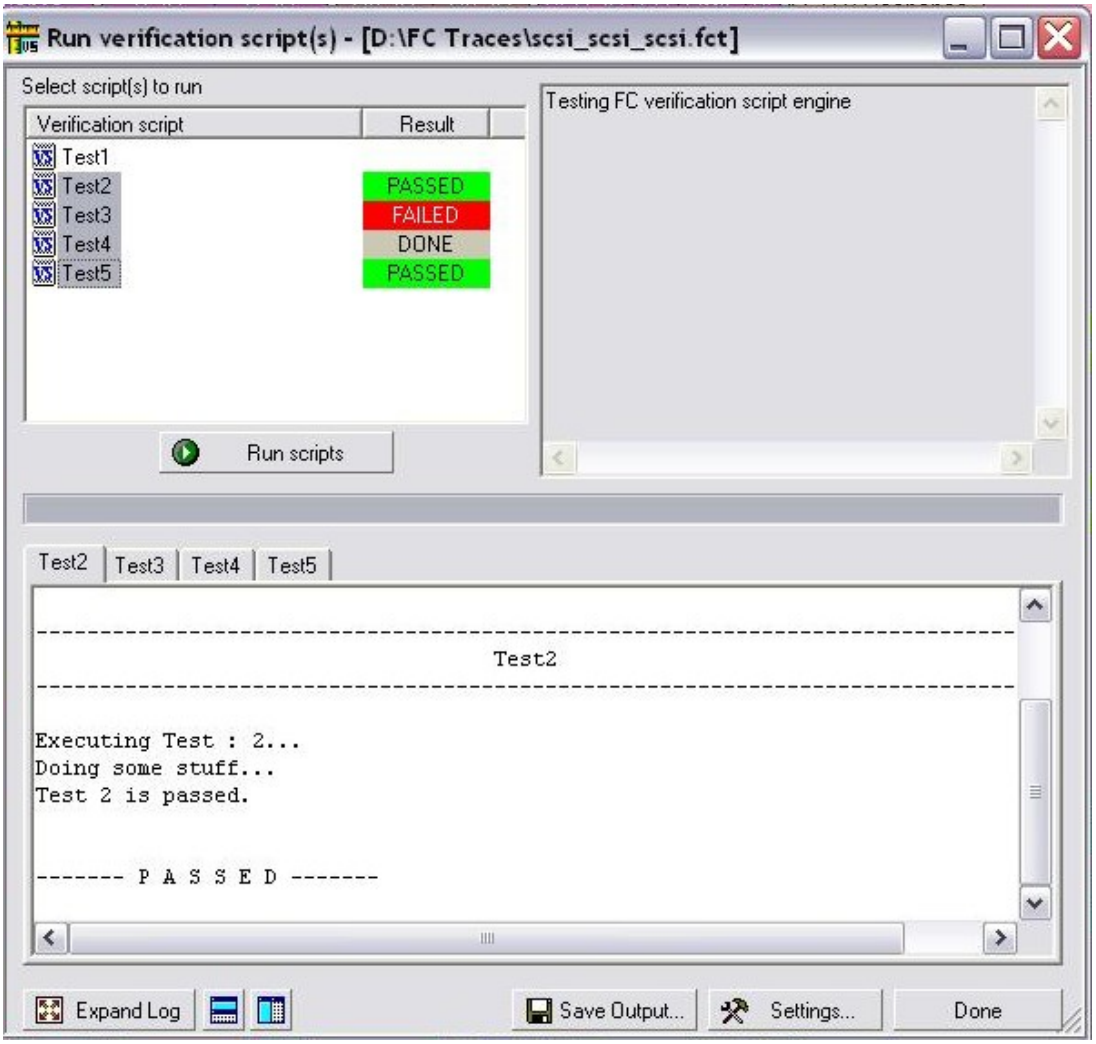


The **Run verification scripts** dialog opens where you choose then run one or several verification scripts:

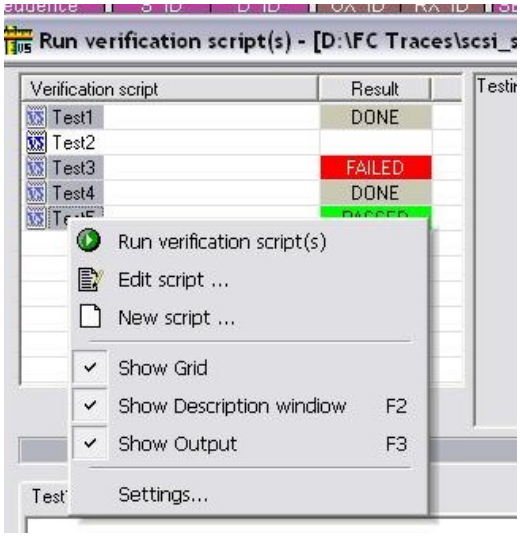


4.1 Running Verification Scripts

Push the button **Run scripts** after you select scripts to run. VSE starts running the selected verification scripts, shows script report information in the output windows, and presents the results of verifications in the script list:



Right-clicking the script list displays some additional operations over selected scripts:



Run verification script(s): Start running selected script(s).

Edit script: Edit selected scripts in the editor application specified in Editor settings.

New script: Create a new script file using the template specified in Editor settings.

Show Grid: Show/hide a grid in the verification script list.

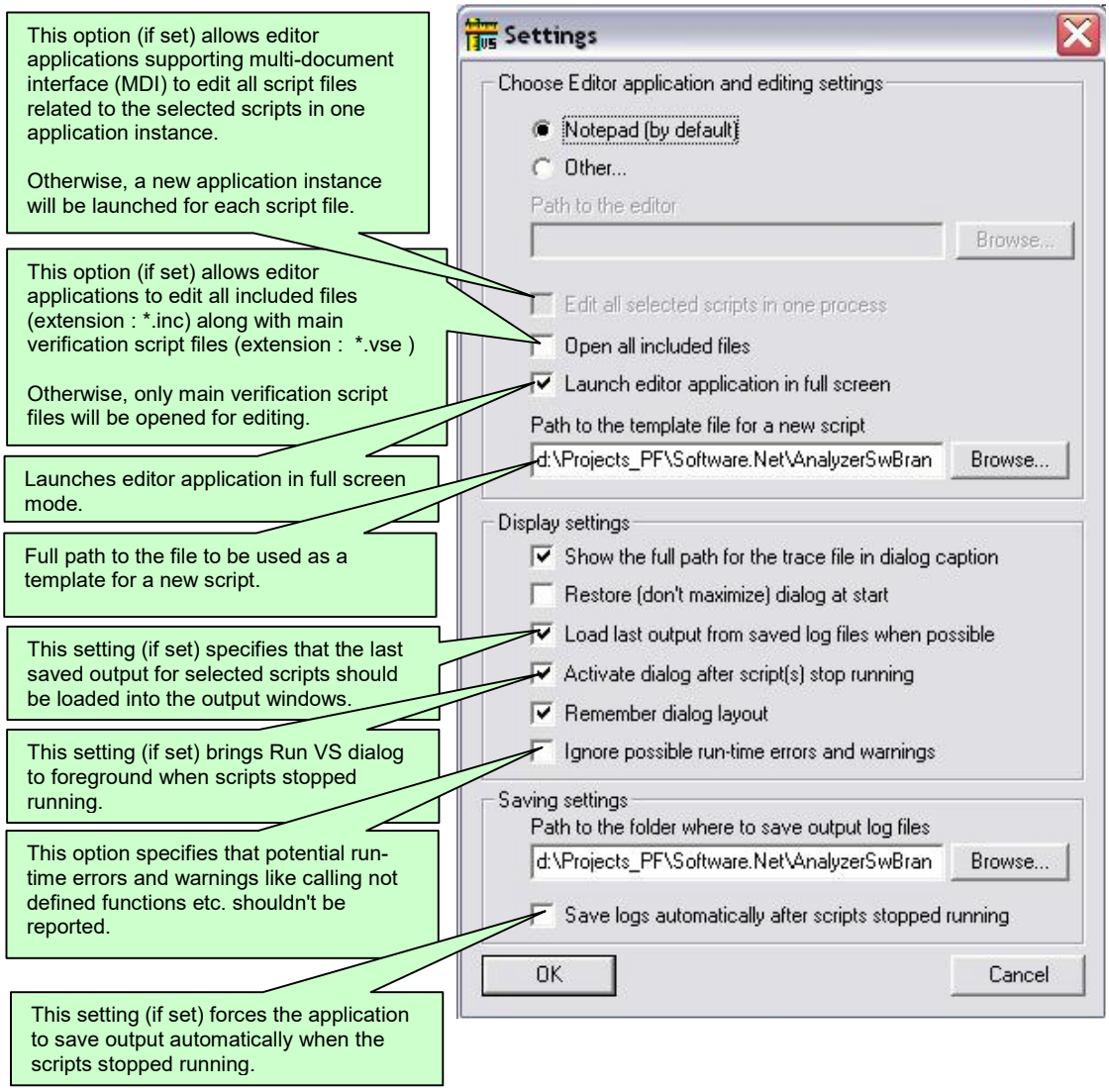
Show Description window: Show/hide the script description window (Shortcut key F2).

Show Output: Show/hide the script output windows (Shortcut key F3).

Settings: Open a special Setting dialog to specify different settings for VSE.

4.2 VSE GUI Settings

After choosing **Settings**, the following dialog appears:



See screen pop-up tooltips for explanation of other settings…

5 Verification Script Engine Input Context Members

All verification scripts have input contexts, special structures that can be used inside of the scripts. The application fills their members. The verification script input contexts have two sets of members:

- Trace event-independent set of members
- Trace event-dependent set of members

Note: All input context members have integer type values unless their types are explicitly specified.
If possible values for input context members are not explicitly specified, they should comply with the current USB specifications.

5.1 Trace Event-independent Set of Members

This set of members is defined and can be used for any event passed to a script:

in.Level: Transaction level of the trace event. It can be one of the following constants:
_PKT: Packet level (value = 0)
_TRA: Transaction level (value = 1)
_SPL_TRA: Split Transaction level (value = 2)
_XFER: Transfer level (value = 3)
_SCSI: SCSI Operation level (value =11)
_PHY_TRA: PHY Transaction level (value = 15)

in.Index: Index of the event in the trace file (packet number for packets etc).

in.Time: Time of the event (type = list, having format = 2 sec 125 ns -> [2 , 125]. For more information, see section 10.1 VSE Time Object.

in.Channel: Indicates on which channel the event occured. It can be one of the following constants:
_USB2: USB2 traffic (value = 1)
_USB2_1: USB2 Channel 1 traffic (value = 2) (Second channel on platferms that support 2 channels)
_USB2_2: USB2 Channel 2 traffic (value = 3)
_USB2_3: USB2 Channel 3 traffic (value = 4)
_USB3_RX: USB3 Host Receive traffic (Upstream, value = 5)
_USB3_TX: USB3 Host Transmit traffic (Downstream, value = 6)
_USB4_HS_ANALYZER_LEFT: USB4 HS Analyzer Left Port orignated traffic (value = 5)
_USB4_HS_ANALYZER_RIGHT: USB4 HS Analyzer Right Port originated traffic (value = 6)
_USB4_HS_EXERCISER_DUT: USB4 HS Exerciser DUT originated traffic (value = 5)
_USB4_HS_EXERCISER_EXR: USB4 HS Exerciser exerciser originated traffic (value = 6)
_USB4_HS_ANALYZER_LEFT_OR_DUT: USB4 HS Analyzer Left Port or Exerciser DUT orignated traffic (value = 5)
_USB4_HS_ANALYZER_RIGHT_OR_EXR: USB4 HS Analyzer Right Port or Exerciser execiser originated traffic (value = 6)
_USB_PD_0: USB Power Delivery traffic (value = 9)
_USB_CC_0: USB Configuration Channel traffic (value = 10)
_USB4_SB_ANALYZER_LEFT: USB4 SB Analyzer Left Port orignated traffic (value = 15)
_USB4_SB_ANALYZER_RIGHT: USB4 SB Analyzer Right Port originated traffic (value = 16)
_USB4_SB_EXERCISER_DUT: USB4 SB Exerciser DUT originated traffic (value = 15)
_USB4_SB_EXERCISER_EXR: USB4 SB Exerciser exerciser originated traffic (value = 16)
_USB4_SB_ANALYZER_LEFT_OR_DUT: USB4 SB Analyzer Left Port or Exerciser DUT orignated traffic (value = 15)

_USB4_SB_ANALYZER_RIGHT_OR_EXR: USB4 SB Analyzer Right Port or Exerciser exerciser originated traffic (value = 16)

in.Speed: Indicates the speed for the trace event. It can be one of the following constants:
_FS: USB2 Full Speed (value = 1)
_LS: USB2 Low Speed (value = 2)
_HS: USB2 High Speed (value = 0) (NOT 3!)
_SS: USB3 Super Speed (value = 4)
_SSP: USB3.1 Super Speed Plus(value = 5)
_USB4_GEN2: USB4 HS Gen2 Speed (value = 8)
_USB4_GEN3: USB4 HS Gen3 Speed (value = 9)
_USB4_SB: USB4 Sideband Speed (value = 10)

in.Direction: Indicates the direction for the trace event. It can be one of the following constants:
_IN: Direction is from Host to Device (value = 2)
_OUT: Direction is from Device to Host (value = 3)
_BOTH: Trace event involves traffic in both directions (value = 4)

in.TraceEvent: Type of trace events. (Application predefined constants are used. See list of possible events in Chapter 7. Verification Script Engine Events.)

in.Notification: Type of notifications. (Application predefined constants are used. Currently no notifications are defined.)

5.2 Trace Event-dependent Set of Members

This set of members is defined and can be used only for a specific event or after calling functions that provide values of variables:

5.2.1 USB Packet-specific Set of Members

Members of this set are valid for all USB packet events:

`in.RawPacket`: Bit source of the whole packet. **Note**: You can extract any necessary information using the **GetNBits()**, **NextNBits()**, or **PeekNBits()** functions. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

`in.RawPacketLength`: Length of the packet (in bytes).

`in.Duration`: Time it took to transmit the packet on the bus (VSE Time object).

`in.Errors`: A bitmap of packet level errors. The table below describes the current list of possible packet errors and their bit positions in the error bitmap:

Note: For USB4, **in.Errors** is not applicable. Use HasPacketError() instead, as shown in Chapter 9.5.

Error type	Bit position	Description
USB2 Errors:		
PID_ERROR	1	Bad PID value
CRC5_ERROR	2	Bad CRC5
CRC16_ERROR	3	Bad CRC16
PACKETLEN_ERROR	4	Wrong packet length
BIT_STUFF_ERROR	5	Bit stuffing error
EOP_ERROR	6	Bad End Of Packet
BABBLE_START_ERROR	7	Babble start error
BABBLE_END_ERROR	8	Babble end (Loss Of Activity) error
FRAME_LENGTH_ERROR	9	Wrong frame length
HANDSHAKE_TIMEOUT_ERROR	10	Bad turnaroud timing or timeout condition for handshake
INTERNAL_ERROR	11	Analyzer internal error
DATA_TOGGLE_ERROR	12	Data toggling error
MICROFRAME_ERROR	13	Bad frame or microframe number
MOD_8_ERROR	14	Last Byte Incomplete error
OTG_ERROR	15	Bad OTG Signal value

Error type	Bit position
USB3 Errors:	

CRC5_ERROR	16
CRC16_ERROR	17
CRC32_ERROR	18
DISPARITY_ERROR	19
10BIT_SYMBOL_ERROR	20
UNKNOWN_PACKET_IPS	21
FRAMING_SYMBOL_ERROR	22
LC_DATA_SYMBOL_ERROR	23
BAD_HEADER_PKT_LENGTH	24
BAD_DATA_LENGTH_FIELD	25
SKP_SYMBOL_ERROR	26
DIR_CTRL_ENDP	27
MISSED_DPH_ERROR	28
MISSED_DPP_ERROR	29
SETUP_DP_ERROR	30
SEQNUM_ERROR	31
PM_LC_TIMEOUT_ERROR	32
PM_ENTRY_TIMEOUT_ERROR	33
Ux_EXIT_TIMEOUT_ERROR	34
NON_ZERO_RSVD_FIELD_ERROR	45
MISSED_ITP_ERROR	46
LATE_EARLY_ITP_ERROR_OBSOLETE	47
SSP_CORRECTABLE_SINGLE_BIT_ERROR	57
ABORTED_DPP_ERROR	69
UNKNOWN_PACKET_USS	74
DEFERRED_DELAYED_ERROR	75
PTP/SCSI Errors:	
PTP_NO_COMMAND_ERROR	35
PTP_NO_RESPONSE_ERROR	36
PTP_ID_DISORDER_ERROR	37
PTP_NO_OPCODE_ERROR	38
SCSI_OP_ERROR_NO_COMMAND	39
SCSI_OP_ERROR_NO_DATA	40
SCSI_OP_ERROR_NO_STATUS	41
SCSI_OP_ERROR_INVALID_OPCODE	42
OBEX_ERROR_NO_STATUS	43
OBEX_ERROR_INVALID_OPCODE	44

PD Errors:	
CRC32_ERROR	48
UNKNOWN_PACKET_ERROR	49
INCOMPLETE_PACKET_ERROR	50
PDO_TYPE_ERROR	51
PDO_VOLTAGE_ERROR	52
PDO_VOLTAGE_ORDER_ERROR	53
PDO_SAME_VOLTAGE_ERROR	54
VENDOR_ID_ERROR	55
STRUCTURED_VDM_VERSION_ERROR	56
TRA_NO_GOODCRC_ERROR	58
TRA_LATE_GOODCRC_ERROR	59
TRA_CORRUPTED_GOODCRC_ERROR	60
NON_ZERO_RSVD_FIELD_ERROR	61
UNK_PKT_NO_PREAMBLE_ERROR	62
UNK_PKT_START_FRAMING_ERROR	63
UNK_PKT_END_FRAMING_ERROR	64
UNK_PKT_PACKET_LEN_ERROR	65
UNK_PKT_SYMBOL_ERROR	66
RESERVED_MESSAGE_TYPE_ERROR	67
INVALID_MESSAGE_TYPE_ERROR	68
XFER_NO_RESPONSE	70
XFER_POWER_NEGO_NO_REQUEST	71
XFER_NO_PS_RDY	72
XFER_NO_SECOND_PS_RDY	73
UNEXPECTED_PORT_DATA_ROLE_ERROR	76
UNEXPECTED_PORT_POWER_ROLE_ERROR	77
OBJECT_POSITION_ERROR	79
OPERATION_CUR_POW_ERROR	80
MIN_MAX_OPERATING_ERROR	81
NUMBER_OF_DATA_OBJECT_ERROR	82
XFER_DATA_RESET_UNSUCCESSFUL	116
VOLTAGE_REGULATION_ERROR	124
TOUCH_TEMP_ERROR	125
SOURCE_INPUT_ERROR	126
PEAK_CURRENT_ERROR	127
EPR_BATTERY_VARIABLE_NOTSUPPORTED	152
DP Errors:	

TRA_NO_REPLY_ERROR	78

Vbus Power parameter values:

in.Power: The value of Vbus power as measured at the time the packet was transmitted (in microwatts). Will be set to a null value if the power information is not present in the trace.

in.Voltage: The value of the Voltage as measured at the time the packet was transmitted (in microvolts). Will be set to a null value if the power information is not present in the trace.

in.Current: The value of the Current as measured at the time the packet was transmitted (in microampers). Will be set to a null value if the power information is not present in the trace.

Skew Measurement parameter values:

in.SkewL1ToL0: Represents the absolute value (in 10 picosecond units) of skew between lanes L0 and L1. A value of 0xFFFFFFFF indicates that no skew measurement corresponding to the packet is available.

in.SkewL1AfterL0: Indicates whether L1 is ahead of or behind L0. A value of 0 means that that L1 is ahead of L0, while a value of 1 means that L0 is ahead of L1. The value is undefined if no skew measurement corresponding to the packet is available (ie, in.SkewL1ToL0 == 0xFFFFFFFF).

5.2.1.1 USB Bus Condition-specific Set of Members

Note: Valid for Bus Condition trace events only, undefined for other events.

`in.BusCondition`: Type of Bus Condition events.

The table below describes the current list of possible Bus Condition events and values of **in.BusCondition**:

Bus Condition	in.BusCondition	Value	Description
CHIRP_K	_CHIRP_K	0	Device "K" Chirp to notify host of Hi Speed capability
CHIRP_J	_CHIRP_J	1	Device "J" Chirp to notify host of Hi Speed capability
FS_K_ON_HS	_FS_K_ON_HS	2	Full Speed "K" signaling on Hi Speed branch
FS_J_ON_HS	_FS_J_ON_HS	3	Full Speed "J" signaling on Hi Speed branch
SUSPEND	_SUSPEND	4	Suspend signaling
RESUME	_RESUME	5	Resume signaling
SE1	_SE1	6	SE1 signaling
SE0	_SE0	7	SE0 FS or LS signaling (can be keep-alive for LS)
VBUS_VOLTAGE_CHANGE	_VBUS_VOLTAGE_CHANGE	44(0x2C)	Change of Vbus Voltage
Reset	_RESET	48(0x30)	Reset signaling

`in.BusCondDuration`: Duration of the Bus Condition event in nanoseconds.

5.2.1.2 USB2 Packet-specific Set of Members

Note: Valid for USB2 packets only, undefined for other events.

The following input context members apply to most of the USB2 packets.

in.Pid: Packet Identifier value for the packet. It is the full PID value, the following constants are defined in the VS_constants.inc include file and can be used by scripts:

```
# USB 2.0 PID values
const PID_OUT   = 0x87;
const PID_IN    = 0x96;
const PID_SOF   = 0xA5;
const PID_SETUP = 0xB4;
const PID_DATA0 = 0xC3;
const PID_DATA1 = 0xD2;
const PID_DATA2 = 0xE1;
const PID_MDATA = 0xF0;
const PID_ACK   = 0x4B;
const PID_NAK   = 0x5A;
const PID_STALL = 0x78;
const PID_NYET  = 0x69;
const PID_PRE   = 0x3C;
const PID_ERR   = 0x3C;
const PID_SPLIT = 0x1E;
const PID_PING  = 0x2D;
const PID_EXT   = 0x0F;
```

in.SubPidPacket: If non-zero, signals that this packet is a protocol extension token as defined by the Link Power Management ECN. In this case the Pid value should be treated as the SubPid (only LPM currently defined).

in.CRC5: CRC5 value for packets (not relevant for Data packets).

5.2.1.2.1 *USB2 Token Packet Members*

in.Addr: Value of the Address (ADDR) field of the Token packet.

in.Endp: Value of the Endpoint (ENDP) field of the Token packet.

5.2.1.2.2 *USB2 Start Of Frame Packet Members*

in.FrmNum: Frame number for the current USB2 frame.

in.MicroFrmNum: Microframe number for the current microframe as determined by the Analyzer (would be zero for Full Speed frames).

5.2.1.2.3 *USB2 Data Packet Members*

in.Payload: Bit source of the packet payload. **Note**: You can extract any necessary information using **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

in.PayloadLength: Length (in bytes) of the packet payload.

in.CRC16: CRC16 value for the Data packet.

5.2.1.2.4 *USB2 Split Token Packet Members*

in.HubAddr: Value of the Hub Address field, the USB device address of the hub supporting the specified full-/low-speed device for this full-/low-speed transaction.

in.SC: Value of the Start/Complete bit for the split token.

in.Port: Value of the Port field – the port number of the target hub for which this full-/low-speed transaction is destined.

in.S: Value of the Speed bit for the split token.

in.E: Value of the End bit for the split token.

in.ET: Value of the Endpoint Type field for the split token.

5.2.1.2.5 *Link Power Management Extended Token Members*

`in.HIRD:` Value of the Host Initiated Resume Duration field.

`in.bLinkState:` Value of the bLinkState field for the LPM extended token.

`in.bRemoteWake:` Value of the bRemoteWake bit for the LPM extended token.

`in.Reserved:` Value of the reserved field for the LPM extended token.

5.2.1.3 USB3 Packet-specific Set of Members

Note 1: Valid for USB3 packets only, undefined for other events.

Note 2: The system does not pass all possible fields of USB3 packets as members of Input Context. If a script needs to use the values of those fields, use the GetHexPktField and GetDecodedPktField functions. See 5.2.1.3.12 for examples.

in.LinkWidth:

const LINK_WIDTH_X1 = 0
const LINK_WIDTH_X2 = 1
const LINK_WIDTH_UNKNOWN = 0xFF

in.LaneNumber: Zero based lane number of the packet. In Bonded mode, it is the lane number of the first symbol.

5.2.1.3.1 *TSEQ Training Sequence Ordered Set Members*

in.Count: Value of TSEQ count, the number of sequential TSEQ sequences sent

5.2.1.3.2 *TS1/TS2 Training Sequence Ordered Set Members*

in.Count: Value of TS count, the number of sequential TS1 or TS2 sequences sent

in.LnkFunctionality: Value of the whole Link Functionality (Link Configuration) byte in the training sequence

in.Reset: Value of the Reset bit in the Link Configuration byte.

in.SSDisable: Value of the Spread Spectrum bit in the Link Configuration byte.

in.Loopback: Value of the Loopback asserted/deasserted bit in the Link Configuration byte.

in.ScrDisable: Value of the Disable Scrambling bit in the Link Configuration byte.

5.2.1.3.3 *LFPS Packet Members*

in.Type: Type of LFPS signaling

in.Duration: Duration of LFPS signaling in nanoseconds

in.DurationSec: Duration of LFPS signaling in second excluding the portion less than 1 second. Should be used together with in.DurationNS. For example returns 3 for duration of 3.845 sec.

in.DurationNS: Duration of LFPS signaling in nanoseconds excluding the portion greater than or equal 1 second. Should be used together with in.DurationSec. For example returns 845000000 for duration of 3.845 sec.

in.PatternType: Type of the pattern that LFPS belongs to. Possible values are:

```
const _NO_PATTERN       = 0x00;
const _SCD1             = 0x01;
const _SCD2             = 0x02;
const _PHY_CAP_5        = 0x03;
const _PHY_CAP_10       = 0x04;
const _PHY_READY        = 0x05;
```

in.StartsPattern: This flag is set when the this packet is first in the pattern it belongs to.

5.2.1.3.4 *Electrical Idle Packet Members*

in.Duration: Duration of Electrical Idle signaling in nanoseconds

in.DurationSec: Duration of Electrical Idle signaling in second excluding the portion less than 1 second. Should be used together with in.DurationNS. For example returns 3 for duration of 3.845 sec.

in.DurationNS: Duration of Electrical Idle signaling in nanoseconds excluding the portion greater than or equal 1 second. Should be used together with in.DurationSec. For example returns 845000000 for duration of 3.845 sec.

in.PatternType: Type of the pattern that Electrical Idle belongs to. Possible values are:

```
const _NO_PATTERN       = 0x00;
const _SCD1             = 0x01;
const _SCD2             = 0x02;
const _PHY_CAP_5        = 0x03;
const _PHY_CAP_10       = 0x04;
const _PHY_READY        = 0x05;
```

in.StartsPattern: This flag is set when the packet is first in the pattern it belongs to.

5.2.1.3.5 *Inter-Packet Symbols Packet Members*

in.NumOfSymbols: Number of symbols in the IPS sequence

5.2.1.3.6 *Link Command Packet Members*

in.LnkCmd: Index of the Link Command type (constants are defined in **VS_constants.inc**)

in.LinkCmdInfo: Value of the Link Command Information field for the Link Command Word

in.CRC5: Value of CRC5 for the Link Command Word

in.LinkCmdInfo_2: Value of the Link Command Information field for the replica of the Link Command Word

in.CRC5_2: Value of CRC5 for the replica of the Link Command Word

5.2.1.3.7 *SKIP Packet Members*

in.Count: Count of symbols in the SKIP sequence

5.2.1.3.8 *Logical Idle Packet Members*

in.Count: Count of symbols in the Logical Idle sequence

5.2.1.3.9 *Link Management Packet Members*

in.SubType: SubType value for LMP. Use constant definitions in **VS_constants.inc.**

5.2.1.3.10 *Isochronous Timestamp Packet Members*

in.BusICounter: Bus Interval Counter value for ITP, in 1/8 of a millisecond

in.Delta: Time Delta value for ITP

in.BusIAdjCtrl: Bus Interval Adjustment Control value for ITP, specifying the device in control of bus interval adjustment

5.2.1.3.11 *Transaction Packet Members*

in.SubType: SubType value for Transaction Packet. Use constant definitions in **VS_constants.inc.**

in.Addr: Address of the device that receives this Transaction Packet

in.Endp: Endpoint number on the device that receives this Transaction Packet

in.Dir: Direction for the specified Endpoint

in.StreamID: Value of the StreamID field

5.2.1.3.12 *Data Packet Members*

in.Addr: Address of the device which is the recipient of this Data Packet

in.Endp: Endpoint number on the device which is the recipient of this Data Packet

in.Dir: Direction for the specified Endpoint

in.StreamID: Value of the StreamID field

in.SeqN: Value of the Sequence Number for this Data Packett

in.DataLength: Length of the payload for this Data Packet

in.PayloadLength: Length of the payload actually recorded for this Data Packet

in.Payload: Bit source of the packet payload for this Data Packet. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

in.CRC32: Value of CRC32 protecting the payload for this Data Packet

Note: In USB Protocol Suite software versions earlier than 3.71 (namely, 3.60 and 3.70), Data Packet Header and Data Packet Payload were two separate packet types with input context members split accordingly between them. Those were combined into a single Data Packet starting with version 3.71.

5.2.1.3.13 *Getting Values of the Fields not present in Input Context*

As mentioned earlier, not all the possible fields of USB3 packets are passed as members of Input Context. The following are examples of using GetHexPktField and GetDecodedPktField functions to get values of the fields not present in Input Context.

```
If( in.TraceEvent == _USB3_LMP_PKT )
{
    # Example of using decoded packet information for LMPs.

    # 'SubType' field,
    # String value
    str = FormatEx("\tSubType(str) = '%s'", GetDecodedPktField("SubType"));
    ReportText( str );

    # Hex Value
    val = GetHexPktField ( "SubType" );
    str = FormatEx( "\tSubType(hex) = 0x%X\n", val );
    ReportText( str );

    # 'Hseq' field
    # String value
    str = FormatEx( "\tHseq(str) = '%s'", GetDecodedPktField ( "Hseq" ) );
    ReportText( str );

    # Hex Value
    val = GetHexPktField ( "Hseq" );
    str = FormatEx( "\tHseq(hex) = 0x%X\n", val );
    ReportText( str );
}

if( ( in.TraceEvent == _USB3_TP_PKT ) && ( in.SubType == TP_ACK ) )
{
    # Example of using decoded packet information for TPs.

    # 'SeqN' field
    # String value
    str = FormatEx( "\tSeqN(str) = '%s'", GetDecodedPktField ( "SeqN" ) );
    ReportText( str );

    # Hex Value
    val = GetHexPktField ( "SeqN" );
    str = FormatEx( "\tSeqN(hex) = 0x%X\n", val );
    ReportText( str );

    # 'NumP' field
    # String value
    str = FormatEx( "\tNumP(str) = '%s'", GetDecodedPktField ( "NumP" ) );
    ReportText( str );


    # Hex Value
```

```
    val = GetHexPktField ( "NumP" );
    str = FormatEx( "\tNumP(hex) = 0x%X\n", val );
    ReportText( str );
}

if( in.TraceEvent == _USB3_DP_PKT )
{
    # Example of using decoded packet information for DPs.

    # 'SeqN' field
    # String value
    str = FormatEx( "\tSeqN(str) = '%s'", GetDecodedPktField ( "SeqN" ) );
    ReportText( str );

    # Hex Value
    val = GetHexPktField ( "SeqN" );
    str = FormatEx( "\tSeqN(hex) = 0x%X\n", val );
    ReportText( str );

    # 'ENDP' field
    # String value
    str = FormatEx( "\tENDP(str) = '%s'", GetDecodedPktField ( "ENDP" ) );
    ReportText( str );

    # Hex Value
    val = GetHexPktField ( "ENDP" );
    str = FormatEx( "\tENDP(hex) = 0x%X\n", val );
    ReportText( str );
}

# Link Control Word
if( ( in.TraceEvent == _USB3_TP_PKT ) ||
    ( in.TraceEvent == _USB3_DP_PKT ) )
{
    ReportText( "LCW:" );
    val = GetHexPktField ( "Hseq" );
    str = FormatEx( "\tHseq = %d", val );
    ReportText( str );

    val = GetHexPktField ( "Hdepth" );
    str = FormatEx( "\tHDepth = %d", val );
    ReportText( str );

    val = GetHexPktField ( "D1" );
    str = FormatEx( "\tDelayed (D1) = %d", val );
    ReportText( str );

    val = GetHexPktField ( "D2" );
    str = FormatEx( "\tDeferred (D2) = %d\n", val );
    ReportText( str );
}
```

5.2.1.4 USB4 Sideband Packet-specific Set of Members

Note: Valid for USB4 SB packets only, undefined for other events.

5.2.1.4.1 *USB4 SB Packet Link Parameters Members*

in.PacketPortRole: Value of the PacketPortRole of the Token Packet's Header. Value **in.PacketPortRole** = 1 means Host, **in.PacketPortRole** = 2 means Device.

5.2.1.4.2 *USB4 SB LT Command Members*

in.LSESymbol: Value of the LSESymbol field of the Token command.

in.ELT: Value of the ELT field of the Token command.

in.Rsvd: Value of the Reserved field of the Token command.

in.LSELane: Value of the LSELane field of the Token command.

in.StartLT: Value of the StartLT field of the Token command.

5.2.1.4.3 *USB4 SB ELT Command Members*

in.Rsvd: Value of the Reserved field of the Token command.

in.ELT: Value of the ELT field of the Token command.

in.ELTtype: Value of the ELTtype field of the Token command.

in.StartLT: Value of the StartLT field of the Token command.

5.2.1.4.4 *USB4 SB AT Command Members*

in.CmdNotResp: Value of the CmdNotResp field of the Token command.

in.ReturnBounce: Value of the ReturnBounce field of the Token command.

in.Recipient: Value of the Recipient field of the Token command.

in.Bounce: Value of the Bounce field of the Token command.

in.Responder: Value of the Responder field of the Token command.

in.Rsvd: Value of the Reserved field of the Token command.

in.StartAT: Value of the StartAT field of the Token command.

in.REG: Value of the Register field of the Token command.

in.LEN: Value of the Data Len field of the Token command.

in.WnR: Value of the READ/WRITE field of the Token command.

in.RegisterDataLen: Size of array **in.RegisterData** of the Token command

in.RegisterData: Array that contains register data of the Token command. If **in.RegisterDataLen** = 0, then the Token command has no register data.

Note: SB AT Command may contain AT/RT register members, depending on the value of **in.REG** member.

5.2.1.4.5 *USB4 SB RT Command Members*

in.CmdNotResp: Value of the CmdNotResp field of the Token command.

in.RTIndex: Value of the Index field of the Token command.

in.Broadcast: Value of the Broadcast field of the Token command.

in.StartRT: Value of the StartRT field of the Token command.

in.REG: Value of the Register field of the Token command. Valid for Addressed commands only.

in.LEN: Value of the Data Len field of the Token command. Valid for Addressed commands only.

in.WnR: Value of the READ/WRITE field of the Token command. Valid for Addressed commands only.

in.RegisterDataLen: Size of array **in.RegisterData** of the Token command. Valid for Addressed commands only.

in.RegisterData: Array that contains register data of the Token command. If **in.RegisterDataLen** = 0, then the Token command has no register data. Valid for Addressed commands only.

in.SelectedGen: Value of the SelectedGen field of the Token command. Valid for Broadcast commands only.

in.Enable3Tx: Value of the Enable3Tx field. Valid for Broadcast commands only.

in.Enable3Rx: Value of the Enable3Rx field. Valid for Broadcast commands only.

in.Lane1: Value of the Lane1Enabled field of the Token command. Valid for Broadcast commands only.

in.Lane0: Value of the Lane0Enabled field of the Token command. Valid for Broadcast commands only.

in.Speed: Value of the TBT3-CompatibleSpeed field of the Token command. Valid for Broadcast commands only.

in.SSCAlwaysOn: Value of the SSCAlwaysOn field of the Token command. Valid for Broadcast commands only.

in.RS_FEC: Value of the RS_FEC field of the Token command. Valid for Broadcast commands only.

in.USB4: Value of the USB4 field of the Token command. Valid for Broadcast commands only.

Note: SB RT Command may contain AT/RT register members, depending on the value of **in.REG** member.

5.2.1.4.6 *USB4 SB AT/RT VendorID Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0x0.

in.VendorIDLow
in.VendorIDHigh
in.VendorIDRsvd1
in.VendorIDRsvd2

5.2.1.4.7 *USB4 SB AT/RT DeviceID Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0x1.

in.DeviceIDLow
in.DeviceIDHigh
in.DeviceIDRsvd1
in.DeviceIDRsvd2

5.2.1.4.8 *USB4 SB AT/RT USB4Version Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0x2 and **GetUsb4LegacyTBT3Protocol()** returns true.

in.USB4Version
in.USB4VersionRsvd1
in.USB4VersionRsvd2
in.USB4VersionRsvd3

5.2.1.4.9 *USB4 SB AT/RT LRDTuning Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0x7 and **GetUsb4LegacyTBT3Protocol()** returns false.

in.LRDTuningRespReq
in.LRDTuningData0
in.LRDTuningData1
in.LRDTuningData2
in.LRDTuningData3

5.2.1.4.10 *USB4 SB AT/RT LinkConfiguration Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0xC.

```
in.LinkCfgLane0Enabled
in.LinkCfgLane1Enabled
im.LinkCfgAsymmetricTx
im.LinkCfgAsymmetricRx
in.LinkCfgRsvd1
in.LinkCfgLane0ReqEnabled
in.LinkCfgLane1ReqEnabled
in.LinkCfgRsvd2
in.LinkCfgBonding
in.LinkCfgGen3
in.LinkCfgRS_FEC_GEN_2
in.LinkCfgRS_FEC_GEN_3
in.LinkCfgRevision1
in.LinkCfgLegacySpeed
in.LinkCfgGen4Support
in.LinkCfgAsymSupportTx
in.LinkCfgAsymSupportRx
in.LinkCfgAsymReqTx
in.LinkCfgAsymReqRx
in.LinkCfgRsvd3
```

5.2.1.4.11 *USB4 SB AT/RT SBChannelVersion Register Members*

Note: These members are available only for SB AT/RT Commands, that have register data and **in.REG** = 0xF.

```
in.SBChVersionSubVersion
in.SBChVersionMinorVersion
in.SBChVersionMajorVersionLSB
in.SBChVersionMajorVersionMSB
```

5.2.1.4.12 *USB4 SB AT/RT TxFFE Register Members*

The following members are common for TxFFE register of **non-retimer** format and **retimer** format:

```
in.Lane0Req, in.Lane0RxLocked, in.Lane0RxActive, in.Lane0ClockSwitchDone,
in.Lane0NewReq
in.Lane1Req, in.Lane1RxLocked, in.Lane1RxActive, in.Lane1ClockSwitchDone,
in.Lane1NewReq
in.Lane0Setting, in.Lane0ReqDone, in.Lane0TxActive
in.Lane1Setting, in.Lane1ReqDone, in.Lane1TxActive
```

Members of TxFFE register of **non-retimer** format:

```
in.Rsvd1, in.Rsvd2
```

Members of TxFFE register of **retimer** format:

```
in.Lane0FwdSwitchDone, in.RetimerRsvd1
in.Lane1FwdSwitchDone, in.RetimerRsvd2
```

Members of TxFFE register of additional **retimer** data:

```
in.Lane0Req, in.Byte4Rsvd1, in.Lane0RxActive, in.Byte4Rsvd2, in.Lane0NewReq
in.Lane1Req, in.Byte5Rsvd1, in.Lane1RxActive, in.Byte5Rsvd2, in.Lane1NewReq
in.Lane0Setting, in.Byte6Rsvd, in.Lane0TxActive
in.Lane1Setting, in.Byte7Rsvd, in.Lane1TxActive
```

5.2.1.4.13 *USB4 SB AT/RT TxFFEGen4 Register Members*

Layout of this register depends on LinkType.
If LinkType is Symmetric, TxFFEGen4 has the following members:

```
in.G4TxFFEReqRx0, in.G4TxFFERsvdRx0 in.NewRequestRx0
in.G4TxFFEReqRx1, in.G4TxFFERsvdRx1 in.NewRequestRx1
in.G4TxFFESetTx0, in.ReqDoneTx0, in.StartTxFFETx0
in.G4TxFFESetTx1, in.ReqDoneTx1, in.StartTxFFETx1
```

If LinkType is Aymmetric3Tx, TxFFEGen4 has the following members:

```
in.G4TxFFEReqRx0, in.G4TxFFERsvdRx0 in.NewRequestRx0
in.G4TxFFEReqRx1, in.G4TxFFERsvdRx1 in.NewRequestRx1
in.G4TxFFEReqRx2, in.G4TxFFERsvdRx2 in.NewRequestRx2
in.G4TxFFESetTx0, in.ReqDoneTx0, in.StartTxFFETx0
```

If LinkType is Asymmetric3Rx, TxFFEGen4 has the following members:

```
in.G4TxFFEReqRx0, in.G4TxFFERsvdRx0 in.NewRequestRx0
in.G4TxFFESetTx0, in.ReqDoneTx0, in.StartTxFFETx0
in.G4TxFFESetTx1, in.ReqDoneTx1, in.StartTxFFETx1
in.G4TxFFESetTx2, in.ReqDoneTx2, in.StartTxFFETx2
```

5.2.1.4.14 *USB4 SB IPS Packet Members*

`in.Packet_Len:` Value of the Packet Length field of the Token Packet.

`in.Data<N>`: Value of the Nth byte of the Token Packet.
For example: `in.Data0`, `in.Data1`, etc.

Note: Number of Data-members is equal to `in.Packet_Len`. For example, Packet with `in.Packet_Len` = 4 has `in.Data0`, `in.Data1`, `in.Data2`, `in.Data3`.

5.2.1.5 USB4 High Speed Packet-specific Set of Members

Note: Valid for USB4 HS packets only, undefined for other events.

5.2.1.5.1 *USB4 HS Packet Link Parameters Members*

Note: Valid for all USB4 HS packets, undefined for other events.

in.LinkWidth: Value of the Link Width of the Token Packet's Header. Value **in.LinkWidth** = 0 means x1 width, **in.LinkWidth** = 1 means x2, **in.LinkWidth** = 2 means x3.

in.IsBonded: Value of the bonding parameter of the Token Packet's Header. Value **in.IsBonded** = 0 means that Lanes are unbonded, **in.IsBonded** = 1 means that Lanes are bonded.

in.LaneNumber: Value of the Lane Number of the Token Packet's Header. Value **in.LaneNumber** = 0 means Lane 0, **in.LaneNumber** = 1 means Lane 1, **in.LaneNumber** = 2 means Lane 2.

5.2.1.5.2 *USB4 HS TLP Packet Header Members*

in.Header_HEC: Value of the HEC field of the Token Packet's Header.

in.Header_Length: Value of the Length field of the Token Packet's Header.

in.Header_HopID: Value of the HopID field of the Token Packet's Header.

in.Header_SuppID: Value of the SuppID field of the Token Packet's Header.

in.Header_PDF: Value of the PDF field of the Token Packet's Header.

5.2.1.5.3 *USB4 HS Control Packet Members*

Note: Control Packet Members include all members of USB4 HS TLP Packet Header.

in.RouteStringLow_TopologyID: Value of the least significant 32 bits of the TopologyID field of the Token Control Packet.

in.RouteStringHigh_TopologyID: Value of the most significant 24 bits of the TopologyID field of the Token Control Packet.

in.RouteStringHigh_Rsvd: Value of the Reserved field of the Route String High field of the Token Control Packet.

in.RouteStringHigh_CM: Value of the CM field of the Route String High field of the Token Control Packet.

in.CRC: Value of CRC protecting the payload for the Token Control Packet.

5.2.1.5.4 *USB4 HS Read/Write Packet Members*

Note: Read/Write Packet Members include all members of USB4 HS Control Packet.

in.Address: Value of the Address field of the Token Packet.

in.DataSize: Value of the DataSize field of the Token Packet.

in.AdapterNum: Value of the AdapterNum field of the Token Packet.

in.CS: Value of the CS field of the Token Packet.

in.SN: Value of the SN field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

5.2.1.5.5 *USB4 HS Read Response/Write Request Packet Members*

Note: Read Response/Write Request Packet Members include all members of USB4 HS Read/Write Packet Members.

in.Data<N>: Value of the Nth byte of Data field of the Token Packet.
For example: **in.Data0**, **in.Data1**, etc.

Note: Number of Data-members is equal to **in.DataSize** * 4 (Number of bytes in DW). For example, Packet with **in.DataSize** = 1 has **in.Data0**, **in.Data1**, **in.Data2**, **in.Data3**.

The byte numbers reflect their position within Transport Layer data structures and not the byte order as transmitted (seen in Figure 4-20 of the USB4 specification). For example, in.Data0 represents DWORD 0 bits 7:0.

5.2.1.5.6 *USB4 HS SLOS1/SLOS2 Packet Members*

in.DataSize: Value of the DataSize field of the Token Packet.

in.Data: Array of bytes representing packet data:
(**in.Data[0]**,
in.Data[1],
…,
in.Data[in.DataSize -1]).

in.Rsvd1: Value of the Rsvd1 field of the Token Packet.

in.SN: Value of the SN field of the Token Packet.

in.Rsvd2: Value of the Rsvd2 field of the Token Packet.

5.2.1.5.7 *USB4 HS Ordered Sets Members*

Note: This category belongs to such Ordered Sets: CL1_REQ, CL2_REQ, CL1_ACK, CL2_ACK, CL0s_ACK, CL_NACK, CL_OFF, TSN, DESKEW, SKIP.

`in.IsScrambled`: Boolean value that indicates whether or not the Ordered Set contents are scrambled.

`in.DataSize`: Size of packet data in bytes

`in.Data`: Array of bytes representing packet data:
(**`in.Data[0]`**,
`in.Data[1]`,
…,
`in.Data[in. DataSize -1]`).

5.2.1.5.8 *USB4 HS CL_WAKE1/CL_WAKE2 Packet Members*

`in.RetimerIndex`: Value of the RetimerIndex field of the Token Packet.

`in.DataSize`: Size of the Data field in bytes of the Token Packet.

`in.Data<N>`: Value of the Nth byte of Data field of the Token Packet.
For example: **`in.Data0`**, **`in.Data1`**, etc.

Note: Number of Data-members is equal to **`in.DataSize`**. For example, Packet with **`in.DataSize`** = 4 has **`in.Data0`**, **`in.Data1`**, **`in.Data2`**, **`in.Data3`**.

The byte numbers reflect their position within Transport Layer data structures and not the byte order as transmitted (seen in Figure 4-20 of the USB4 specification). For example, in.Data0 represents DWORD 0 bits 7:0.

5.2.1.5.9 *USB4 HS CL_WAKE1-2 Packet Members*

`in.DataSize`: Size of the Data field in bytes of the Token Packet.

`in.Data<N>`: Value of the Nth byte of Data field of the Token Packet.
For example: **`in.Data0`**, **`in.Data1`**, etc.

Note: Number of Data-members is equal to **`in.DataSize`**. For example, Packet with **`in.DataSize`** = 4 has **`in.Data0`**, **`in.Data1`**, **`in.Data2`**, **`in.Data3`**.

The byte numbers reflect their position within Transport Layer data structures and not the byte order as transmitted (seen in Figure 4-20 of the USB4 specification). For example, in.Data0 represents DWORD 0 bits 7:0.

in.CLWake1RetimerIndexes: List of CL_WAKE1 Retimer Indexes. **in.CLWake1RetimerIndexes[N]** – N[th] CL_WAKE1 Retimer Index (**in.CLWake1RetimerIndexes[0]**, **in.CLWake1RetimerIndexes[1]**, …, **in.CLWake1RetimerIndexes[in.CLWake1RetimerIndexesSize-1]**).

in.CLWake1RetimerIndexesSize: Number of the CL_WAKE1 Retimer Indexes.

in.CLWake2RetimerIndexes: List of CL_WAKE1 Retimer Indexes. **in.CLWake2RetimerIndexes[N]** – N[th] CL_WAKE2 Retimer Index (**in.CLWake2RetimerIndexes[0]**, **in.CLWake2RetimerIndexes[1]**, …, **in.CLWake2RetimerIndexes[in.CLWake2RetimerIndexesSize-1]**).

in.CLWake2RetimerIndexesSize: Number of the CL_WAKE2 Retimer Indexes.

5.2.1.5.10 *USB4 HS TS1/TS2 Packet Members*

in.TSID: Value of the TSID field of the Token Packet.

in.Rsvd0: Value of the Rsvd0 field of the Token Packet.

in.LBT2: Value of the LBT2 field of the Token Packet.

in.Rsvd1: Value of the Rsvd1 field of the Token Packet.

in.Rsvd2: Value of the Rsvd2 field of the Token Packet.

in.TSLaneNumber: Value of the Lane Number field of the Token Packet.

in.LBT: Value of the LBT field of the Token Packet.

in.Rsvd3: Value of the Rsvd3 field of the Token Packet.

in.DataSize: Size of packet data in bytes

in.Data: Array of bytes representing packet data:
(**in.Data[0]**,
in.Data[1],
…,
in.Data[in. DataSize -1]).

5.2.1.5.11 *USB4 HS LFPS Packet Members*

in.State: Value of the State field of the Token Packet.

5.2.1.5.12 *USB4 HS Follow-Up Packet Members*

Note: Follow-Up Packet Members include all members of USB4 HS TLP Packet Header.

in.RequestTSLo: Value of the least significant 32 bits of the RequestTS field of the Token Packet.

in.RequestTSMid: Value from the 32nd to 63rd bits of the RequestTS field of the Token Packet.

in.RequestTSHi: Value of the most significant 16 bits of the RequestTS field of the Token Packet.

in.ResponseTSLo: Value of the least significant 32 bits of the ResponseTS field of the Token Packet.

in.ResponseTSMid: Value from the 32nd to 63rd bits of the ResponseTS field of the Token Packet.

in.ResponseTSHi: Value of the most significant 16 bits of the ResponseTS field of the Token Packet.

in.TimeOffsetFromHRLo: Value of the least significant 32 bits of the TimeOffsetFromHR field of the Token Packet.

in.TimeOffsetFromHRHi: Value of the most significant 32 bits of the TimeOffsetFromHR field of the Token Packet.

in.FreqOffsetFromHR: Value of the FreqOffsetFromHR field of the Token Packet.

in.SleepCyclesN: Value of the SleepCyclesN field of the Token Packet.

in.S2U: Value of the S2U field of the Token Packet.

in.IDTimeStampLo: Value of the least significant 32 bits of the IDTimeStamp field of the Token Packet.

in.IDTimeStampMid: Value from the 32nd to 63rd bits of the IDTimeStamp field of the Token Packet.

in.IDTimeStampHi: Value of the most significant 16 bits of the IDTimeStamp field of the Token Packet.

in.TimeOffsetFromInterDomainHRLo: Value of the least significant 32 bits of the TimeOffsetFromInterDomainHR field of the Token Packet.

in.TimeOffsetFromInterDomainHRHi: Value of the most significant 32 bits of the TimeOffsetFromInterDomainHR field of the Token Packet.

in.FreqOffsetFromInterDomainHR: Value of the FreqOffsetFromInterDomainHR field of the Token Packet.

in.CRC32: Value of CRC32 protecting the payload for the Token Packet.

5.2.1.5.13 *USB4 HS Inter-Domain Time Stamp Packet Members*

Note: Inter-Domain Time Stamp Packet Members include all members of USB4 HS TLP Packet Header.

in.Rsvd: Value of the Reserved field of the Token Packet.

in.IDTimeStampLo: Value of the least significant 32 bits of the IDTimeStamp field of the Token Packet.

in.IDTimeStampMid: Value from the 32nd to 63rd bits of the IDTimeStamp field of the Token Packet.

in.IDTimeStampHi: Value of the most significant 16 bits of the IDTimeStamp field of the Token Packet.

in.TimeOffsetFromInterDomainGMLo: Value of the least significant 32 bits of the TimeOffsetFromInterDomainGM field of the Token Packet.

in.TimeOffsetFromInterDomainGMHi: Value of the most significant 32 bits of the TimeOffsetFromInterDomainGM field of the Token Packet.

in.FreqOffsetFromInterDomainGM: Value of the FreqOffsetFromInterDomainGM field of the Token Packet.

in.CRC32: Value of CRC32 protecting the payload for the Token Packet.

5.2.1.5.14 *USB4 HS Credit Grant Packet Members*

Note: Credit Grant Packet Members include all members of USB4 HS TLP Packet Header.

in.Records_Number: Value of the RecordCount field of the Token Packet.

Note: All members of Credit Grant Packet below are indexed. Index is indicated by suffix **<N>** and depends on the Record number starting from 0. For example, **in.ECC<N>** means that the Packet has Members **in.ECC0**, **in.ECC1**, etc.

in.ECC<N>: Value of the ECC field of the Nth Record of the Token Packet.

in.Credits<N>: Value of the Credits field of the Nth Record of the Token Packet.

in.CreditHopID<N>: Value of the CreditHopID field of the Nth Record of the Token Packet.

in.Rsvd<N>: Value of the Reserved field of the Nth Record of the Token Packet.

in.LFlag<N>: Value of the L Flag field of the Nth Record of the Token Packet.

5.2.1.5.15 *USB4 HS Path Credit Sync Packet Members*

Note: Path Credit Sync Packet Members include all members of USB4 HS TLP Packet Header.

in.ECC: Value of the ECC field of the Token Packet.

in.PCC: Value of the PCC field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

5.2.1.5.16 *USB4 HS Shared Buffers Credit Sync Packet Members*

Note: Shared Buffers Credit Sync Packet Members include all members of USB4 HS TLP Packet Header.

in.ECC: Value of the ECC field of the Token Packet.

in.SCC: Value of the SCC field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

5.2.1.5.17 *USB4 HS Notification Packet Members*

Note: Notification Packet Members include all members of USB4 HS Control Packet.

in.EventCode: Value of the EventCode field of the Token Packet.

in.AdapterNum: Value of the AdapterNum field of the Token Packet. The field is left for compatibility.

in.EventInfo: Value of the EventInfo field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

in.PG: Value of the PG field of the Token Packet.

Note: The following fields are actual if **in.EventCode** is equal to 36 (DP_CON_CHANGE event):

in.ConnectorNum: Value of the ConnectorNum sub-field of the EventInfo field of the Token Packet.

in.IRQ_HPD: Value of the IRQ_HPD sub-field of the EventInfo field of the Token Packet.

in.PlugUnplug: Value of the PlugUnplug sub-field of the EventInfo field of the Token Packet.

5.2.1.5.18 *USB4 HS Notification Acknowledgement Packet Members*

Note: Notification Acknowledgement Packet Members include all members of USB4 HS Control Packet.

5.2.1.5.19 *USB4 HS Hot Plug Event Packet Members*

Note: Hot Plug Event Packet Members include all members of USB4 HS Control Packet.

in.AdapterNum: Value of the AdapterNum field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

in.UPG: Value of the UPG field of the Token Packet.

5.2.1.5.20 *USB4 HS Tunneled Packet Members*

Note: Tunneled Packet Members include all members of USB4 HS TLP Packet Header.

in.TunneledData: Bit source of the packet's tunneled payload. **Note**: You can extract any necessary information using **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

in.TunneledDataLen: Length (in bytes) of the packet's tunneled payload.

in.Data<N>: Value of the Nth byte of Data field of the Token Packet.
For example: **in.Data0**, **in.Data1**, etc.

Note: Number of Data-members is equal to **in.Header_Length**. For example, Packet with **in.Header_Length** = 4 has **in.Data0**, **in.Data1**, **in.Data2**, **in.Data3**.

The byte numbers reflect their position within Transport Layer data structures and not the byte order as transmitted (seen in Figure 4-20 of the USB4 specification). For example, in.Data0 represents DWORD 0 bits 7:0.

5.2.1.5.21 *USB4 HS E2E Credit Grant Packet Members*

Note: E2E Credit Grant Packet Members include all members of USB4 HS TLP Packet Header.

in.ECC: Value of the ECC field of the Token Packet.

in.Credits: Value of the Credits field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

in.CS: Value of the CS field of the Token Packet.

5.2.1.5.22 *USB4 HS E2E Credit Sync Packet Members*

Note: E2E Credit Sync Packet Members include all members of USB4 HS TLP Packet Header.

in.ECC: Value of the ECC field of the Token Packet.

in.Credits: Value of the Credits field of the Token Packet.

in.Rsvd: Value of the Reserved field of the Token Packet.

in.CS: Value of the CS field of the Token Packet.

5.2.1.5.23 *USB4 HS IPS/USS/START_RS_FEC Packet Members*

in.Packet_Len: Value of the Packet Length field of the Token Packet.

in.Data<N>: Value of the N[th] byte of the Token Packet.
For example: **in.Data0**, **in.Data1**, etc.

Note: Number of Data-members is equal to **in.Packet_Len**. For example, Packet with **in.Packet_Len** = 4 has **in.Data0**, **in.Data1**, **in.Data2**, **in.Data3**.

The byte numbers reflect their position within Transport Layer data structures and not the byte order as transmitted (seen in Figure 4-20 of the USB4 specification). For example, in.Data0 represents DWORD 0 bits 7:0.

5.2.1.5.24 *USB4 HS Filtered Packet Members*

in.Type: Value of the origin packet type of the Token Packet.

in.Count: Value of the OS Count of the Token Packet.

5.2.1.5.25 *USB4 Gen 4 Training Sequences Members*

Note: Gen 4 Training Sequences: TS1, TS2, TS3, TS4.

in.Cursor: Value of the Cursor field of the Token Packet.

in.Indication: Value of the Indication field of the Token Packet.

in.IndicationComplement: Value of the Indication Complement field of the Token Packet.

in.Counter: Value of the Counter field of the Token Packet.

in.CounterComplement: Value of the Counter Complement field of the Token Packet.

Note: TS2.clksw does not have any specific member.

5.2.1.5.26 *USB4 Gen 4 Ordered Sets Members*

Note: Gen 4 Ordered Sets: DESKEW, TSNOS, IGNORE, UNBOND, CL0s_EXIT, CL_OFF.

5.2.1.5.26.1 **USB4 Gen 4 DESKEW Packet Members**

in.DeskewIndex: Value of the Index field of the Token Packet.

5.2.1.5.26.2 **USB4 Gen 4 TSNOS Packet Members**

in.SymbolsDelay: Value of the Symbols Delay field of the Token Packet.

5.2.1.5.26.3 **USB4 Gen 4 CL0s_EXIT Packet Members**

in.CL0sPhase: Value of the CL0sPhase of the Token Packet.

5.2.1.5.26.4 USB4 Gen 4 CL_OFF Packet Members

in.CLOffIndex: Value of the Index field of the Token Packet.

in.CLState: Value of the CLState field of the Token Packet.

5.2.1.6 USB4 HS Inter-Domain Packet-specific Set of Members

Note: Valid for USB4 HS Inter-Domain packets only, undefined for other events.

5.2.1.6.1 *USB4 HS Inter-Domain Request/Response Packet Members*

Note: Inter-Domain Request/Response Packet Members include all members of USB4 HS Control Packet, USB4 HS Inter-Domain Packet Header and USB4 HS Inter-Domain Packet Payload .

in.DataSize: Value of the DataSize field of the Token Packet.

in.Rsvd1: Value of the Rsvd1 field of the Token Packet.

in.SN: Value of the SN field of the Token Packet.

in.Rsvd2: Value of the Rsvd2 field of the Token Packet.

in.IDPProtocolType: Inter-Domain protocol type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_HS_IDP_DISCOVERY	Inter-Domain Discovery
_USB4_HS_IDP_USB4NET	Inter-Domain USB4NET

5.2.1.6.2 *USB4 HS Inter-Domain Discovery Packet Members*

in.ProtocolUUID<N>: Value of the Nth DWORD of Inter-Domain Discovery Protocol UUID field of the Token Packet's IDP Header.
For example: **in. ProtocolUUID0, in.ProtocolUUID1**, etc.

in.IDPPacketType: Value of the Inter-Domain Discovery packet type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_HS_IDP_UUID_REQUEST	UUID Request
_USB4_HS_IDP_UUID_RESPONSE	UUID Response
_USB4_HS_IDP_PROPERTIES_READ_REQUEST	Properties Read Request
_USB4_HS_IDP_PROPERTIES_READ_RESPONSE	Properties Read Response
_USB4_HS_IDP_PROPERTIES_CHANGED_NOTIFICATION	Properties Changed Notification

_USB4_HS_IDP_PROPERTIES_CHANGED_RESPONSE	Properties Changed Response
_USB4_HS_IDP_ERROR_RESPONSE	Error Response
_USB4_HS_IDP_LINK_STATE_STATUS_REQUEST	Link State Status Request
_USB4_HS_IDP_LINK_STATE_STATUS_RESPONSE	Link State Status Response
_USB4_HS_IDP_LINK_STATE_CHANGE_REQUEST	Link State Change Request
_USB4_HS_IDP_LINK_STATE_CHANGE_RESPONSE	Link State Change Response
Any other value	Reserved

`in.IDPPayloadDataLen:` Count of IDP Payload Data bytes of the Token Packet.

`in.IDPPayloadData`: List of IDP Payload Data bytes. **`in.IDPPayloadData[N]`** – Nth IDP Payload Data byte of the Token Packet.
(**`in.IDPPayloadData[0]`**,
`in.IDPPayloadData[1]`,
…,
`in.IDPPayloadData[in.IDPPayloadDataLen-1]`).

Note: **`in.IDPPayloadData`** provides access to packet data after Inter-Domain Discovery Header data.

5.2.1.6.3 *USB4 HS Inter-Domain Discovery UUID Request Packet Members*

Note: Inter-Domain UUID Request Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

5.2.1.6.4 *USB4 HS Inter-Domain Discovery UUID Response Packet Members*

Note: Inter-Domain UUID Response Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

`in.SourceUUID<N>`: Value of the Nth DWORD of Source UUID field of the Token Packet.
For example: **`in.SourceUUID0`**, **`in.SourceUUID1`**, etc.

`in.HasSourceRoute`: Indicates whether the Token Packet has Source Route field.

`in.SourceRoute<N>:` Value of the Nth DWORD of Source Route field of the Token Packet.
For example: **`in.SourceRoute0`**, **`in.SourceRoute1`**.

5.2.1.6.5 *USB4 HS Inter-Domain Discovery Properties Read Request Packet Members*

Note: Inter-Domain Properties Read Request Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.SourceUUID<N>: Value of the N[th] DWORD of Source UUID field of the Token Packet.
For example: **in.SourceUUID0**, **in.SourceUUID1**, etc.

in.DestinationUUID<N>: Value of the N[th] DWORD of Destination UUID field of the Token Packet.
For example: **in.DestinationUUID0**, **in.DestinationUUID1**, etc.

in.Offset: Value of the Offset field of the Token Packet.

in.Rsvd3: Value of the Rsvd3 field of the Token Packet.

5.2.1.6.6 *USB4 HS Inter-Domain Discovery Properties Read Response Packet Members*

Note: Inter-Domain Properties Read Response Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.SourceUUID<N>: Value of the N[th] DWORD of Source UUID field of the Token Packet.
For example: **in.SourceUUID0**, **in.SourceUUID1**, etc.

in.DestinationUUID<N>: Value of the N[th] DWORD of Destination UUID field of the Token Packet.
For example: **in.DestinationUUID0**, **in.DestinationUUID1**, etc.

in.Offset: Value of the Offset field of the Token Packet.

in.PropertiesBlockSize: Value of the Properties Block Size field of the Token Packet.

in.PropertiesBlockGeneration: Value of the Properties Block Generation field of the Token Packet.

in.PropertiesBlockDataLen: Count of Properties Block Data bytes of the Token Packet.

in.PropertiesBlockData: List of Properties Block Data bytes. **in.PropertiesBlockData[N]** – N[th] Properties Block Data byte of the Token Packet.
(**in.PropertiesBlockData[0]**,
in.PropertiesBlockData[1],
…,
in.PropertiesBlockData[in.PropertiesBlockDataLen-1]).

5.2.1.6.7 *USB4 HS Inter-Domain Discovery Properties Changed Notification Packet Members*

Note: Inter-Domain Properties Changed Notification Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.SourceUUID<N>: Value of the N[th] DWORD of Source UUID field of the Token Packet.
For example: **in.SourceUUID0**, **in.SourceUUID1**, etc.

5.2.1.6.8 *USB4 HS Inter-Domain Discovery Error Response Packet Members*

Note: Inter-Domain Error Response Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.ErrorResponse: Value of the Error Response field of the Token Packet.

5.2.1.6.9 *USB4 HS Inter-Domain Discovery Link State Status Request Packet Members*

Note: Inter-Domain Link State Status Request Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

5.2.1.6.10 *USB4 HS Inter-Domain Discovery Link State Status Response Packet Members*

Note: Inter-Domain Link State Status Response Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.Status: Value of the Status field of the Token Packet.

in.SLW: Value of the SLW field of the Token Packet.

in.TLW: Value of the TLW field of the Token Packet.

in.SLS: Value of the SLS field of the Token Packet.

in.TLS: Value of the TLS field of the Token Packet.

5.2.1.6.11 *USB4 HS Inter-Domain Discovery Link State Change Request Packet Members*

Note: Inter-Domain Link State Change Request Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.TLW: Value of the TLW field of the Token Packet.

in.TLS: Value of the TLS field of the Token Packet.

in.Rsvd3: Value of the Rsvd3 field of the Token Packet.

5.2.1.6.12 *USB4 HS Inter-Domain Discovery Link State Change Response Packet Members*

Note: Inter-Domain Link State Change Response Packet Members include all members of USB4 HS Inter-Domain Discovery Packet.

in.Status: Value of the Status field of the Token Packet.

5.2.1.6.13 *USB4 HS Inter-Domain USB4NET Packet Members*

in.ProtocolUUID<N>: Value of the Nth DWORD of Inter-Domain USB4NET Protocol UUID field of the Token Packet's IDP Header.
For example: **in. ProtocolUUID0, in.ProtocolUUID1**, etc.

in.RequestorUUID<N>: Value of the Nth DWORD of Inter-Domain USB4NET Requestor UUID field of the Token Packet's IDP Header.
For example: **in.RequestorUUID0, in.RequestorUUID1**, etc.

in.ResponderUUID<N>: Value of the Nth DWORD of Inter-Domain USB4NET Responder UUID field of the Token Packet's IDP Header.
For example: **in.ResponderUUID0, in.ResponderUUID1**, etc.

in.RequestID: Value of RequestID field of the Token Packet.

in.IDPPacketType: Value of the Inter-Domain USB4NET packet type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_HS_IDP_LOGIN_REQUEST	Login Request
_USB4_HS_IDP_LOGIN_RESPONSE	Login Response
_USB4_HS_IDP_LOGOUT_REQUEST	Logout Request
_USB4_HS_IDP_LOGOUT_RESPONSE	Logout Response
Any other value	Reserved

in.IDPPayloadDataLen: Count of IDP Payload Data bytes of the Token Packet.

in.IDPPayloadData: List of IDP Payload Data bytes. **in.IDPPayloadData[N]** – Nth IDP Payload Data byte of the Token Packet.
(**in.IDPPayloadData[0]**,
in.IDPPayloadData[1],
…,
in.IDPPayloadData[in.IDPPayloadDataLen-1]).

Note: **in.IDPPayloadData** provides access to packet data after Inter-Domain USB4NET Header data.

5.2.1.6.14 *USB4 HS Inter-Domain USB4NET Login Resquest Packet Members*

Note: Inter-Domain Login Request Packet Members include all members of USB4 HS Inter-Domain USB4NET Packet.

in.Usb4NetServiceRevision: Value of the Usb4NetServiceRevision field of the Token Packet.

in.ServiceHopId: Value of the ServiceHopId field of the Token Packet.

in.Rsvd3: Value of the 3rd Reserved field of the Token Packet.

in.Rsvd4: Value of the 4th Reserved field of the Token Packet.

in.Rsvd5: Value of the 5th Reserved field of the Token Packet.

in.Rsvd6: Value of the 6th Reserved field of the Token Packet.

5.2.1.6.15 *USB4 HS Inter-Domain USB4NET Login Response Packet Members*

Note: Inter-Domain Login Response Packet Members include all members of USB4 HS Inter-Domain USB4NET Packet.

in.Status: Value of the Status field of the Token Packet.

in.ResponderMACAddressLo: Value of the least significant 32 bits of the ResponderMACAddress field of the Token Packet.

in.ResponderMACAddressHi: Value of the most significant 32 bits of the ResponderMACAddress field of the Token Packet.

in.ResponderMACAddressLength: Value of the ResponderMACAddressLength field of the Token Packet.

in.Rsvd3: Value of the 3rd Reserved field of the Token Packet.

in.Rsvd4: Value of the 4th Reserved field of the Token Packet.

in.Rsvd5: Value of the 5th Reserved field of the Token Packet.

in.Rsvd6: Value of the 6th Reserved field of the Token Packet.

5.2.1.6.16 *USB4 HS Inter-Domain USB4NET Logout Request Packet Members*

Note: Inter-Domain Logout Request Packet Members include all members of USB4 HS Inter-Domain USB4NET Packet.

5.2.1.6.17 *USB4 HS Inter-Domain USB4NET Logout Response Packet Members*

Note: Inter-Domain Logout Response Packet Members include all members of USB4 HS Inter-Domain USB4NET Packet.

`in.Status`: Value of the Status field of the Token Packet.

5.2.1.7 USB4 HS Tunneled DP Main Link Packet-specific Set of Members

Note: Valid for USB4 HS Tunneled DP Main Link packets only, undefined for other events.
Note: To get access to given set of members use function SetUsb4TunProtocol().

Note: Tunneled DP Main Link Set of Members include all members of USB4 HS Tunneled Packet.

in.TunDPLinkWidth: Value of the DP Link Width of the Token Tunneled Packet. Value **in.TunDPLinkWidth** = 1 means x1 width, **in.TunDPLinkWidth** = 2 means x2, **in.TunDPLinkWidth** = 4 means x4. This value depends on the Tunneled Protocol constant passed to SetUsb4TunProtocol() (_USB4_TUN_DP_X1, _USB4_TUN_DP_X2 or _USB4_TUN_DP_X4).

in.TunDPPacketType: Tunneled DP Main Link packet type of the Token Tunneled Packet. May be one of the values in table below.

Packet type	Description
_USB4_TUN_DP_VIDEO_DATA_PKT	SST Video Data Packet
_USB4_TUN_DP_BLANK_START_PKT	SST Blank Start Packet
_USB4_TUN_DP_MSA_PKT	SST Main Stream Attribute Packet
_USB4_TUN_DP_SDP_PKT	SST Secondary Stream Packet
_USB4_TUN_DP_CLOCK_SYNC_PKT	DP Clock Sync Packet
_USB4_TUN_DP_MULTI_STREAM_PKT	Multi Stream Packet
_USB4_TUN_DP_FEC_DECODE_PKT	FEC Decode Packet
_USB4_TUN_DP_LINKCTRL_ALPM_PKT	DP Link Control ALPM Packet

5.2.1.7.1 *TU Set Based Packets*

Note: This category describes common members for SST Video Data Packet, SST Secondary Stream Packet and Multi Stream Packet.

Note: TU Set Based Packet Members include all members of Tunneled DP Main Link Packet.

in.TUSetsCount: Count of TU Sets in the Token Packet.

in.TUSet<N>DataLen: Count of Data Symbols of the Nth TU Set in the Token Packet.
For example: **in.TUSet0DataLen**, **in.TUSet1DataLen**, etc.

in.TUSet<N>Data: List of Data Symbols of the Nth TU Set in the Token Packet.
For example: **in.TUSet0Data**, **in.TUSet1Data**, etc. Number of Data Symbols in **in.TUSet<N>Data** is equal to **in.TUSet<N>DataLen**. Supports indexing (for example, **in.TUSet0Data[0]** = value of the first Data Symbol in TU Set 0.)

5.2.1.7.2 *SST Video Data Packet*

Note: SST Video Data Packet Members include all members of TU Set Based Packet.
in.TUSet<N>ECC: ECC field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>VideoCount: Video Count field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>FillCount: Fill Count field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>EOCIndex: EOC Index field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>L: L field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>Rsvd: Reserved field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>EOC: EOC field of TU Header of the Nth TU Set in the Token Packet.

5.2.1.7.3 *SST Blank Start Packet*

Note: SST Blank Start Packet Members include all members of Tunneled DP Main Link Packet.

in.ECC: ECC field of the Blank Start Header of the Token Packet.

in.FillCount: Fill Count field of the Blank Start Header of the Token Packet.

in.Rsvd: Reserved field of the Blank Start Header of the Token Packet.

in.CP: CP field of the Blank Start Header of the Token Packet.

in.SR: SR field of the Blank Start Header of the Token Packet.

in.VB_ID_0: The 1st VB_ID field of the payload of the Token Packet.

in.VB_ID_1: The 2nd VB_ID field of the payload of the Token Packet.

in.VB_ID_2: The 3rd VB_ID field of the payload of the Token Packet.

in.VB_ID_3: The 4th VB_ID field of the payload of the Token Packet.

in.Mvid70_0: The 1st Mvid 7:0 field of the payload of the Token Packet.

in.Mvid70_1: The 2nd Mvid 7:0 field of the payload of the Token Packet.

in.Mvid70_2: The 3rd Mvid 7:0 field of the payload of the Token Packet.

in.Mvid70_3: The 4th Mvid 7:0 field of the payload of the Token Packet.

in.Maud70_0: The 1st Maud 7:0 field of the payload of the Token Packet.

in.Maud70_1: The 2nd Maud 7:0 field of the payload of the Token Packet.

in.Maud70_2: The 3rd Maud 7:0 field of the payload of the Token Packet.

in.Maud70_3: The 4th Maud 7:0 field of the payload of the Token Packet.

5.2.1.7.4 *SST Main Stream Attribute Packet*

Note: SST Main Stream Attribute Packet Members include all members of Tunneled DP Main Link Packet.

in.ECC: ECC field of the MSA Header of the Token Packet.

in.FillCount: Fill Count field of the MSA Header Header of the Token Packet.

in.Rsvd: Reserved field of the MSA Header of the Token Packet.

in.Mvid0: The 1st Mvid field of the payload of the Token Packet.

in.Mvid1: The 2nd Mvid field of the payload of the Token Packet.

in.Mvid2: The 3rd Mvid field of the payload of the Token Packet.

in.Mvid3: The 4th Mvid field of the payload of the Token Packet.

in.Nvid: Nvid field of the payload of the Token Packet.

in.Htotal: Htotal field of the payload of the Token Packet.

in.Hstart: Hstart field of the payload of the Token Packet.

in.Hwidth: Hwidth field of the payload of the Token Packet.

in.Vtotal: Vtotal field of the payload of the Token Packet.

in.Vstart: Vstart field of the payload of the Token Packet.

in.Vheight: Vheight field of the payload of the Token Packet.

in.HSP_HSW: HSP_HSW field of the payload of the Token Packet.

in.VSP_VSW: VSP_VSW field of the payload of the Token Packet.

in.MISC0: MISC0 field of the payload of the Token Packet.

in.MISC1: MISC1 field of the payload of the Token Packet.

5.2.1.7.5 *SST Secondary Stream Packet*

Note: SST Secondary Stream Packet Members include all members of TU Set Based Packet.

in.TUSet<N>ECC: ECC field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>SecondaryCount: SecondaryCount field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>FillCount: FillCount field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>L: L field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>NSE: NSE field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>NSS: NSS field of TU Header of the Nth TU Set in the Token Packet.

in.TUSet<N>ND: ND field of TU Header of the Nth TU Set in the Token Packet.

5.2.1.7.6 *DP Clock Sync Packet*

Note: DP Clock Sync Packet Members include all members of Tunneled DP Main Link Packet.

in.Rsvd0: The 1st Reserved field of the Token Packet.

in.Rsvd1: The 2nd Reserved field of the Token Packet.

in.WindowCount: WindowCount field of the Token Packet.

in.FLCLo: Value of the least significant 32 bits of FLC field of the Token Packet.

in.FLCMid: Value from the 32nd to 63rd bits of FLC field of the Token Packet.

in.FLCHi: Value of the most significant 8 bits of FLC field of the Token Packet.

in.Rsvd2: The 3rd Reserved field of the Token Packet.

in.Rsvd3: The 4th Reserved field of the Token Packet.

in.CRC32: Value of CRC32 protecting the payload of the Token Packet.

5.2.1.7.7 *Multi Stream Packet*

Note: Multi Stream Packet Members include all members of TU Set Based Packet.

in.TUSet<N>ECC: ECC field of Sub-MTP TU Header of the Nth Sub-MTP TU Set in the Token Packet.

in.TUSet<N>DataCount: DataCount field of Sub-MTP TU Header of the Nth Sub-MTP TU Set in the Token Packet.

in.TUSet<N>Type: Type field of Sub-MTP TU Header of the Nth Sub-MTP TU Set in the Token Packet.

in.TUSet<N>SlotNumber: SlotNumber field of Sub-MTP TU Header of the Nth Sub-MTP TU Set in the Token Packet.

in.TUSet<N>Parameter: Parameter field of the Nth Sub-MTP TU Set in the Token Packet.

5.2.1.7.8 *FEC Decode Packet*

Note: FEC Decode Packet Members include all members of Tunneled DP Main Link Packet.

in.Command0SRCount: SRCount field of the 1st FEC Command of the Token Packet.

in.Command0FDS: FDS field of the 1st FEC Command of the Token Packet.

in.Command0FEN: FEN field of the 1st FEC Command of the Token Packet.

in.Rsvd0: The 1st Reserved field of the Token Packet.

in.Command1SRCount: SRCount field of the 2nd FEC Command of the Token Packet.

in.Command1FDS: FDS field of the 2nd FEC Command of the Token Packet.

in.Command1FEN: FEN field of the 2nd FEC Command of the Token Packet.

in.Rsvd1: The 2nd Reserved field of the Token Packet.

in.Command2SRCount: SRCount field of the 3rd FEC Command of the Token Packet.

in.Command2FDS: FDS field of the 3rd FEC Command of the Token Packet.

in.Command2FEN: FEN field of the 3rd FEC Command of the Token Packet.

5.2.1.7.9 *DP Link Control ALPM Packet*

Note: DP Link Control ALPM Packet Members include all members of Tunneled DP Main Link Packet.

in.SW: SW field of the Token Packet.

in.SRCount: SRCount field of the Token Packet.

in.Type: Type field of the Token Packet.

5.2.1.8 USB4 HS Tunneled DP AUX Packet-specific Set of Members

Note: Valid for USB4 HS Tunneled DP AUX packets only, undefined for other events.
Note: To get access to given set of members use function SetUsb4TunProtocol().

Note: Tunneled DP AUX Set of Members include all members of USB4 HS Tunneled Packet.

in.TunDPAuxPacketType: Tunneled DP AUX packet type of the Token Tunneled Packet. May be one of the values in table below.

Packet type	Description
_USB4_TUN_DPAUX_AUX_PKT	AUX Packet
_USB4_TUN_DPAUX_HDP_STATUS_PKT	HDP Status Packet
_USB4_TUN_DPAUX_SET_CONFIG_PKT	SET_CONFIG Packet
_USB4_TUN_DPAUX_ACK_PKT	ACK Packet

5.2.1.8.1 *AUX Packet*

Note: AUX Packet Members include all members of Tunneled DP AUX Packet.

in.DPAuxPacketDirection: Tunneled DP AUX packet direction of the Token AUX Packet. May be one of the values in table below.

Direction	Description
_USB4_TUN_DPAUX_DIRECTION_REQUESTER	Display Port Requester
_USB4_TUN_DPAUX_DIRECTION_REPLIER	Display Port Replier

in.CRC32: Value of CRC32 protecting the payload of the Token AUX Packet.

in.DataLen: Size in bytes of Data field of the Token AUX Packet.

in.Data: Data field of the Token AUX Packet.

5.2.1.8.2 *AUX Request Packet*

Note: AUX Request Packet Members include all members of AUX Packet.

in.RequestCommand: Type of AUX Request Command of the Token Packet. May be one of the values in table below.

Request command	Description
_USB4_TUN_DPAUX_REQ_I2C_WRITE	I^2C Write
_USB4_TUN_DPAUX_REQ_I2C_READ	I^2C Read

_USB4_TUN_DPAUX_REQ_I2C_WSUR	I²C Write_Status_Update_Request
_USB4_TUN_DPAUX_REQ_I2C_MOT_WRITE	I²C MOT Write
_USB4_TUN_DPAUX_REQ_I2C_MOT_READ	I²C MOT Read
_USB4_TUN_DPAUX_REQ_I2C_MOT_WSUR	I²C MOT Write_Status_Update_Request
_USB4_TUN_DPAUX_REQ_WRITE	Write
_USB4_TUN_DPAUX_REQ_READ	Read

in.RequestAddress: Address field of the Token AUX Request Packet.

in.RequestDataLen: DataLen field of the Token AUX Request Packet.

5.2.1.8.3 *AUX Reply Packet*

Note: AUX Reply Packet Members include all members of AUX Packet.

in.ReplyCommand: Type of AUX Reply Command of the Token Packet. May be one of the values in table below.

Reply command	Description
_USB4_TUN_DPAUX_REP_ACK	ACK
_USB4_TUN_DPAUX_REP_NACK	AUX NACK
_USB4_TUN_DPAUX_REP_DEFER	AUX DEFER
_USB4_TUN_DPAUX_REP_I2C_NACK	I²C NACK
_USB4_TUN_DPAUX_REP_I2C_DEFER	I²C DEFER

5.2.1.8.4 *HDP Status Packet*

Note: HDP Status Packet Members include all members of Tunneled DP AUX Packet.

in.ECC: ECC field of the Token Packet.

in.Rsvd: Reserved field of the Token Packet.

in.Plug: Plug field of the Token Packet.

5.2.1.8.5 *SET_CONFIG Packet*

Note: SET_CONFIG Packet Members include all members of Tunneled DP AUX Packet.

in.ECC: ECC field of the Token Packet.

in.LinkRate: LinkRate field of the Token Packet.

in.LinkRate = 0 represents 1.62 GHz

in.LinkRate = 1 represents 2.70 GHz
in.LinkRate = 2 represents 5.40 GHz
in.LinkRate = 3 represents 8.10 GHz

in.LaneCount: LaneCount field of the Token Packet.
in.LaneCount = 0 represents LinkDown
in.LaneCount = 1 represents 1 Lane
in.LaneCount = 2 represents 2 Lanes
in.LaneCount = 4 represents 4 Lanes

in.MessageType: MessageType field of the Token Packet.

in.MessageData: MessageData field of the Token Packet.

5.2.1.8.6 *ACK Packet*

Note: ACK Packet Members include all members of Tunneled DP AUX Packet.

in.ECC: ECC field of the Token Packet.

in.ACKType: ACKType field of the Token Packet.

in.Rsvd: Reserved field of the Token Packet.

5.2.1.9 USB4 HS Tunneled PCIe Packet-specific Set of Members

Note: Valid for USB4 HS Tunneled PCIe packets only, undefined for other events.
Note: To get access to given set of members use function SetUsb4TunProtocol().

Note: Tunneled PCIe Set of Members include all members of USB4 HS Tunneled Packet.

in.TunPCIePacketType: Tunneled PCIe packet type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_TUN_PCIE_ORDERED_SET	Ordered Set
_USB4_TUN_PCIE_EIS	Electrical Idle State Packet
_USB4_TUN_PCIE_TLP	TLP Packet
_USB4_TUN_PCIE_DLLP	DLLP Packet
_USB4_TUN_PCIE_PERST_ACTIVE	PERST Active
_USB4_TUN_PCIE_PERST_INACTIVE	PERST Incative
_USB4_TUN_PCIE_TLP_NEXT	TLP Continuation
_USB4_TUN_PCIE_TLP_NULL	Nullified TLP
_USB4_TUN_PCIE_UNKNOWN	Unknown

5.2.1.9.1 *USB4 HS Tunneled PCIe Ordered Sets*

Note: USB4 HS Tunneled PCIe Ordered Sets Members include all members of Tunneled PCIe Packet.

in.OrderedSetType: Tunneled PCIe Ordered Set type of the Token Packet. May be one of the values in table below.

OS type	Description
_USB4_TUN_PCIE_EIOS	Electrical Idle Ordered Set
_USB4_TUN_PCIE_TS1	TS1
_USB4_TUN_PCIE_TS2	TS2
_USB4_TUN_PCIE_OS_INVALID	Invalid Ordered Set

5.2.1.9.2 *USB4 HS Tunneled PCIe EIOS*

Note: EIOS Members include all members of Tunneled PCIe Ordered Sets.

in.EIOS_COMSymbol: EIOS_COMSymbol field of the Token Packet.

in.EIOS_IDLSymbols: EIOS_IDLSymbols field of the Token Packet.

5.2.1.9.3 *USB4 HS Tunneled PCIe TS1/TS2 Packet*

Note: TS1/TS2 Packet Members include all members of Tunneled PCIe Ordered Sets.

in.TS_COMSymbol: TS_COMSymbol field of the Token Packet.

in.TS_LinkNumber: TS_LinkNumber field of the Token Packet.

in.TS_LaneNumber: TS_LaneNumber field of the Token Packet.

in.TS_N_FTS: TS_N_FTS field of the Token Packet.

in.TS_TrainingControl: TS_TrainingControl field of the Token Packet.

in.TS_DataRate: TS_DataRate field of the Token Packet.

in.TS_RawSymbols: List of TS1/TS2 Symbols of the Token Packet. The list contains _USB4_TUN_PCIE_TS_SYMBOLS_COUNT (10) elements (**in.TS_RawSymbols[0]**, **in.TS_RawSymbols[1]**, …, **in.TS_RawSymbols[_USB4_TUN_PCIE_TS_SYMBOLS_COUNT-1]**).

5.2.1.9.4 *USB4 HS Tunneled PCIe TLP/DLLP Packets*

Note: USB4 HS Tunneled PCIe TLP/DLLP Packet Members include all members of Tunneled PCIe Packet.

Tunneled TLP/DLLP may contain multiple packets, so every field has index of inner packet, which starts with zero. The following fields are common:

in.TS_ Count: The number of inner packets.

in. IsTlp[i]: Whether an inner packet is a TLP

in.IsDllp[i]: Whether an inner packet is a DLLP.

5.2.1.9.5 *USB4 HS Tunneled PCIe inner TLP Packets*

Every field has index of inner packet, which starts with zero.

in.TLPType[i]: Tunneled PCIe TLP type of the Token Packet. May be one of the values in table below.

TLP type	Description
_USB4_TUN_PCIE_TLP_MRD32	Memory Read Request
_USB4_TUN_PCIE_TLP_MRDLK32	Memory Read Request-Locked
_USB4_TUN_PCIE_TLP_MWR32	Memory Write Request
_USB4_TUN_PCIE_TLP_MRD64	Memory Read Request 64-bit
_USB4_TUN_PCIE_TLP_MRDLK64	Memory Read Request-Locked 64-bit
_USB4_TUN_PCIE_TLP_MWR64	Memory Write Request 64-bit

_USB4_TUN_PCIE_TLP_IORD	I/O Read Request
_USB4_TUN_PCIE_TLP_IOWR	I/O Write Request
_USB4_TUN_PCIE_TLP_CFGRD_0	Configuration Read Type 0
_USB4_TUN_PCIE_TLP_CFGWR_0	Configuration Write Type 0
_USB4_TUN_PCIE_TLP_CFGRD_1	Configuration Read Type 1
_USB4_TUN_PCIE_TLP_CFGWR_1	Configuration Write Type 1
_USB4_TUN_PCIE_TLP_TCFGRD	Trusted Configuration Read (Deprecated)
_USB4_TUN_PCIE_TLP_TCFGWR	Trusted Configuration Write (Deprecated)
_USB4_TUN_PCIE_TLP_MSG	Message Request
_USB4_TUN_PCIE_TLP_MSGD	Message Request with data payload
_USB4_TUN_PCIE_TLP_CPL	Completion without Data
_USB4_TUN_PCIE_TLP_CPLD	Completion with Data
_USB4_TUN_PCIE_TLP_CPLLK	Completion for Locked Memory Read without Data
_USB4_TUN_PCIE_TLP_CPLDLK	Completion for Locked Memory Read
_USB4_TUN_PCIE_TLP_FETCH_ADD	Fetch and Add AtomicOp Request
_USB4_TUN_PCIE_TLP_SWAP	Unconditional Swap AtomicOp Request
_USB4_TUN_PCIE_TLP_CAS	Compare and Swap AtomicOp Request
_USB4_TUN_PCIE_TLP_FETCH_ADD64	Fetch and Add AtomicOp Request 64-bit
_USB4_TUN_PCIE_TLP_SWAP64	Unconditional Swap AtomicOp Request 64-bit
_USB4_TUN_PCIE_TLP_CAS64	Compare and Swap AtomicOp Request 64-bit
_USB4_TUN_PCIE_TLP_LPRFX	Local TLP Prefix
_USB4_TUN_PCIE_TLP_EPRFX	End-End TLP Prefix
_USB4_TUN_PCIE_TLP_INVALID	Invalid TLP

in.TLP_SeqNum[i]: TLP Sequence Number of the of the Token Packet.

in.TLPHeader_Fmt[i]: Fmt field of the TLP Header of the Token Packet.

in.TLPHeader_Type[i]: Type field of the TLP Header of the Token Packet.

in.TLPHeader_T9[i]: T9 field of the TLP Header of the Token Packet.

in.TLPHeader_TC[i]: TC field of the TLP Header of the Token Packet.

in.TLPHeader_T8[i]: T8 field of the TLP Header of the Token Packet.

in.TLPHeader_Attr[i]: Attr field of the TLP Header of the Token Packet.

in.TLPHeader_LN[i]: LN field of the TLP Header of the Token Packet.

in.TLPHeader_TH[i]: TH field of the TLP Header of the Token Packet.

in.TLPHeader_TD[i]: TD field of the TLP Header of the Token Packet.

in.TLPHeader_EP[i]: EP field of the TLP Header of the Token Packet.

in.TLPHeader_Attr2[i]: Attr2 field of the TLP Header of the Token Packet.

in.TLPHeader_AT[i]: AT field of the TLP Header of the Token Packet.
Value **in.TLPHeader_AT[i]** = 0 means "Untranslated",
in.TLPHeader_AT[i] = 1 means "TranslationRequest",
in.TLPHeader_AT[i] = 2 means "Translated",
in.TLPHeader_AT[i] = 3 means "Reserved".

in.TLPHeader_Length[i]: Length field of the TLP Header of the Token Packet.

in.RequesterID_BusNumber[i]: RequesterID BusNumber field of the of the Token Packet.

in.RequesterID_DeviceNumber[i]: RequesterID DeviceNumber field of the of the Token Packet.

in.RequesterID_FunctionNumber[i]: RequesterID FunctionNumber field of the of the Token Packet.

in.Tag[i]: Tag field of the of the Token Packet. **Note:** There are packets, that have no Tag field in decoding (TLP Vendor Defined Message Packets), they have Reserved field instead. But most of TLP Packets has Tag field, so it is included in common TLP fields.

in.DataLen[i]: Length of Data field in bytes of the Token Packet. If packet has no Data field DataLen = 0.

in.Data[i]: Bit source of Data field Of the Token Packet. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

in.HasCRC32[i]: Presence of CRC32 below in the current packet. May be absent, if PCIe TLP does not fit fully.

in.CRC32[i]: Value of CRC32 protecting the payload of the Token Packet.

in.TLP_CHK[i]: CHK field of the of the Token Packet.

Note: Tunneled PCIe TLP packet can have 0 or more prefixes.

in.PrefixCount[i]: Count of TLP prefixes of the Token Packet.

in.PrefixFmtList[i]: List of Fmt fields of prefixes. **in.PrefixFmtList[i] [N]** – Fmt field of Nth TLP prefix (**in.PrefixFmtList[i][0]**, **in.PrefixFmtList[i][1]**, …, **in.PrefixFmtList[i][in.PrefixCount[i]-1]**).

in.PrefixTypeList[i]: List of Type fields of prefixes. **in.PrefixTypeList[N]** – Type field of Nth TLP prefix (**in.PrefixTypeList[i][0]**, **in.PrefixTypeList[i][1]**, …, **in.PrefixTypeList[i][in.PrefixCount[i]-1]**). May be one of the values in table below.

Prefix type	Description
_USB4_TUN_PCIE_TLP_PT_LOCAL_MR_IOV	MR-IOV TLP Prefix
_USB4_TUN_PCIE_TLP_PT_LOCAL_VEND_PREFIX_L0	Vendor Defined Local TLP Prefix L0

_USB4_TUN_PCIE_TLP_PT_LOCAL_VEND_PREFIX_L1	Vendor Defined Local TLP Prefix L1
_USB4_TUN_PCIE_TLP_PT_END_END_TPH	TPH TLP Prefix
_USB4_TUN_PCIE_TLP_PT_END_END_PASID	PASID
_USB4_TUN_PCIE_TLP_PT_END_END_VEND_PREFIX_E0	Vendor Defined End-End TLP Prefix E0
_USB4_TUN_PCIE_TLP_PT_END_END_VEND_PREFIX_E1	Vendor Defined End-End TLP Prefix E1

`in.PrefixDataList[i]`: List of Data fields of prefixes. **`in.PrefixDataList[i][N]`** – Data field of Nth TLP prefix.

`in.TPHPrefixSTList[i]`: List of ST fields of TPH prefixes. **`in.TPHPrefixSTList[i][N]`** – ST field of Nth TPH TLP prefix.

`in.TPHPrefixRsvdList[i]`: List of Rsvd fields of TPH prefixes. **`in.TPHPrefixRsvdList[i][N]`** – Rsvd field of Nth TPH TLP prefix.

Note: **`in.TPHPrefixSTList[i][N]`** value and **`in.TPHPrefixRsvdList[i][N]`** value are correct for TPH prefix, any other prefix has **`in.PrefixDataList[i][N]`** value. However, to keep correct indexing, each prefix has access to both **`in.TPHPrefixSTList[i][N]`**, **`in.TPHPrefixRsvdList[i][N]`** and **`in.PrefixDataList[i][N].`**

Example:

```
for (j = 0; j < in.PrefixCount[i]; ++j)
{
    cur_prefix_fmt = in.PrefixFmtList[i][j];
    cur_preifx_type_name = GetUsb4TunPCIeTLPPrefixName( in.PrefixTypeList[i][j] );
    if (in.PrefixTypeList[i][j] == _USB4_TUN_PCIE_TLP_PT_END_END_TPH)
    {
        cur_TPHPrefix_ST = in.TPHPrefixSTList[i][j];
        cur_TPHPreifx_Rsvd = in.TPHPrefixRsvdList[i][j];
    }
    else
    {
        cur_prefix_data = in.PrefixDataList[i][j];
    }
}
```

5.2.1.9.6 *USB4 HS Tunneled PCIe TLP Memory and AtomicOp Request Packets*

Note: This category belongs to such TLP Packet types:
Memory Read Request 32-bit and 64-bit,
Memory Read Request-Locked 32-bit and 64-bit,
Memory Write Request 32-bit and 64-bit,
Fetch and Add AtomicOp Request 32-bit and 64-bit,
Unconditional Swap AtomicOp Request 32-bit and 64-bit,
Compare and Swap AtomicOp Request 32-bit and 64-bit.

Note: Packet from this category include all members of Tunneled PCIe TLP Packet.

in.TLPHeader_1stDwBE[i]: 1stDwBE field of the TLP Header of the Token Packet.

in.TLPHeader_LastDwBE[i]: LastDwBE field of the TLP Header of the Token Packet.

in.TLPHeader_PH[i]: PH field of the TLP Header of the Token Packet.
in.TLPHeader_PH[i] = 0 : "Bi-directional data structure";
in.TLPHeader_PH[i] = 1 : "Requester";
in.TLPHeader_PH[i] = 2 : "Target";
in.TLPHeader_PH[i] = 3 : "Target with Priority";

in.TLPHeader_Address[i]: Address field of the TLP Header of the Token Packet. **Valid for 32-bit packets only**.

in.TLPHeader_AddressHi[i]: Value of the most significant DWORD of Address field of the TLP Header of the Token Packet. **Valid for 64-bit packets only**.

in.TLPHeader_AddressLo[i]: Value of the less significant DWORD of Address field of the TLP Header of the Token Packet. **Valid for 64-bit packets only**.

5.2.1.9.7 *USB4 HS Tunneled PCIe TLP I/O Read Write Request Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Packet.

in.TLPHeader_1stDwBE[i]: 1stDwBE field of the TLP Header of the Token Packet.

in.TLPHeader_LastDwBE[i]: LastDwBE field of the TLP Header of the Token Packet.

in.TLPHeader_Address[i]: Address field of the TLP Header of the Token Packet.

in.TLPHeader_Rsvd[i]: Reserved field of the TLP Header of the Token Packet.

5.2.1.9.8 *USB4 HS Tunneled PCIe TLP Configuration Packets*

Note: This category belongs to such TLP Packet types:
Configuration Read Type 0
Configuration Write Type 0
Configuration Read Type 1
Configuration Write Type 1
Trusted Configuration Read (Deprecated)
Trusted Configuration Write (Deprecated)

Note: Packet from this category include all members of Tunneled PCIe TLP Packet.

in.TLPHeader_1stDwBE[i]: 1stDwBE field of the TLP Header of the Token Packet.

in.TLPHeader_LastDwBE[i]: LastDwBE field of the TLP Header of the Token Packet.

in.DeviceID_BusNumber[i]: DeviceID BusNumber field of the of the Token Packet.

in.DeviceID_DeviceNumber[i]: DeviceID DeviceNumber field of the of the Token Packet.

in.DeviceID_FunctionNumber[i]: DeviceID FunctionNumber field of the of the Token Packet.

in.TLPHeader_Rsvd[i]: Reserved field of the TLP Header of the Token Packet.

in.TLPHeader_Register[i]: Register field of the TLP Header of the Token Packet.

5.2.1.9.9 *USB4 HS Tunneled PCIe TLP Completion Packets*

Note: This category belongs to such TLP Packet types:
Completion without Data,
Completion with Data,
Completion for Locked Memory Read without Data,
Completion for Locked Memory Read.

Note: Packet from this category include all members of Tunneled PCIe TLP Packet.

in.CompleterID_BusNumber[i]: CompleterID BusNumber field of the of the Token Packet.

in.CompleterID_DeviceNumber[i]: CompleterID DeviceNumber field of the of the Token Packet.

in.CompleterID_FunctionNumber[i]: CompleterID FunctionNumber field of the of the Token Packet.

in.TLPHeader_ComplStatus[i]: ComplStatus field of the TLP Header of the Token Packet.
in.TLPHeader_ComplStatus[i] = 0 : "Successful Completion";
in.TLPHeader_ComplStatus[i] = 1 : "Unsupported Request";
in.TLPHeader_ComplStatus[i] = 2 : "Configuration Request Retry Status";
in.TLPHeader_ComplStatus[i] = 4 : "Completer Abort";

in.TLPHeader_BCM[i]: BCM field of the TLP Header of the Token Packet.

in.TLPHeader_ByteCount[i]: ByteCount field of the TLP Header of the Token Packet.

in.TLPHeader_LowerAddress[i]: LowerAddress field of the TLP Header of the Token Packet.

in.TLPHeader_Rsvd[i]: Reserved field of the TLP Header of the Token Packet.

5.2.1.9.10 *USB4 HS Tunneled PCIe TLP Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Packet.

in.TLPHeader_MessageCode[i]: Message Code of the Token TLP Message Packet. May be one of the values in table below.

Message Code	Description

_USB4_TUN_PCIE_TLP_MSG_CODE_UNLOCK	Unlock
_USB4_TUN_PCIE_TLP_MSG_CODE_INVALID_REQUEST	Invalid Request
_USB4_TUN_PCIE_TLP_MSG_CODE_INVALID_COMPLETION	Invalid Completion
_USB4_TUN_PCIE_TLP_MSG_CODE_PAGE_REQUEST	Page Request
_USB4_TUN_PCIE_TLP_MSG_CODE_PRG_RESPONSE	PRG Response
_USB4_TUN_PCIE_TLP_MSG_CODE_LTR	Locked Transactions Support
_USB4_TUN_PCIE_TLP_MSG_CODE_OBFF	Optimized Buffer Flush/Fill
_USB4_TUN_PCIE_TLP_MSG_CODE_PM_ACTIVE_STATE_NACK	PM_Active_State_Nack
_USB4_TUN_PCIE_TLP_MSG_CODE_PM_PME	PM_PME
_USB4_TUN_PCIE_TLP_MSG_CODE_PME_TURN_OFF	PME_Turn_Off
_USB4_TUN_PCIE_TLP_MSG_CODE_PME_TO_ACK	PME_TO_Ack
_USB4_TUN_PCIE_TLP_MSG_CODE_ASSERT_INTA	Assert_INTA
_USB4_TUN_PCIE_TLP_MSG_CODE_ASSERT_INTB	Assert_INTB
_USB4_TUN_PCIE_TLP_MSG_CODE_ASSERT_INTC	Assert_INTC
_USB4_TUN_PCIE_TLP_MSG_CODE_ASSERT_INTD	Assert_INTD
_USB4_TUN_PCIE_TLP_MSG_CODE_DEASSERT_INTA	Deassert_INTA
_USB4_TUN_PCIE_TLP_MSG_CODE_DEASSERT_INTB	Deassert_INTB
_USB4_TUN_PCIE_TLP_MSG_CODE_DEASSERT_INTC	Deassert_INTC
_USB4_TUN_PCIE_TLP_MSG_CODE_DEASSERT_INTD	Deassert_INTD
_USB4_TUN_PCIE_TLP_MSG_CODE_ERR_COR	ERR_COR
_USB4_TUN_PCIE_TLP_MSG_CODE_ERR_NONFATAL	ERR_NONFATAL
_USB4_TUN_PCIE_TLP_MSG_CODE_ERR_FATAL	ERR_FATAL
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED1	Ignored1
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED2	Ignored2
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED3	Ignored3
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED4	Ignored4
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED5	Ignored5
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED6	Ignored6
_USB4_TUN_PCIE_TLP_MSG_CODE_IGNORED7	Ignored7
_USB4_TUN_PCIE_TLP_MSG_CODE_SET_SLOT_POWER_LIMIT	Set_Slot_Power_limit
_USB4_TUN_PCIE_TLP_MSG_CODE_PTM_REQUEST	PTMRequest
_USB4_TUN_PCIE_TLP_MSG_CODE_PTM_RESPONSE_RESPONSE_D	PTMResponse_ResponseD
_USB4_TUN_PCIE_TLP_MSG_CODE_VENDOR_DEFINED_TYPE0	Vendor_Defined_Type_0
_USB4_TUN_PCIE_TLP_MSG_CODE_VENDOR_DEFINED_TYPE1	Vendor_Defined_Type_1

5.2.1.9.11 *USB4 HS Tunneled PCIe TLP Message Reserved Packets*

Note: This category belongs to such TLP Message Codes:
Assert_INTA,

Assert_INTB,
Assert_INTC,
Assert_INTD,
Deassert_INTA,
Deassert_INTB,
Deassert_INTC,
Deassert_INTD,
PM_Active_State_Nack,
PM_PME,
PME_Turn_Off,
PME_TO_Ack,
ERR_COR,
ERR_NONFATAL,
ERR_FATAL,
Unlock,
PTMRequest,
PTMResponse_ResponseD (in case of **in.TLPType[i]** = _USB4_TUN_PCIE_TLP_MSG, else – see USB4 HS Tunneled PCIe TLP PTM Response_ResponseD Packets),
Set_Slot_Power_limit.

Note: Packet from this category include all members of Tunneled PCIe TLP Message Packet.

in.TLPHeader_Rsvd0[i]: 1st Reserved field of the TLP Header of the Token TLP Message Packet.

in.TLPHeader_Rsvd1[i]: 2nd Reserved field of the TLP Header of the Token TLP Message Packet.

5.2.1.9.12 *USB4 HS Tunneled PCIe TLP Message LTR Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Message Packet.

in.TLPHeader_Rsvd[i]: Reserved field of the TLP Header of the Token TLP Message Packet.

in.TLPHeader_NoSnoopLatency[i]: NoSnoopLatency field of the TLP Header of the Token TLP Message Packet.

in.TLPHeader_SnoopLatency[i]: SnoopLatency field of the TLP Header of the Token TLP Message Packet.

5.2.1.9.13 *USB4 HS Tunneled PCIe TLP Message OBFF Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Message Packet.

in.TLPHeader_Rsvd0[i]: 1st Reserved field of the TLP Header of the Token TLP Message Packet.

in.TLPHeader_OBFFCode[i]: OBFF Code field of the TLP Header of the Token TLP Message Packet.
- **in.TLPHeader_OBFFCode** = 0 : "Idle";
- **in.TLPHeader_OBFFCode** = 1 : "OBFF";
- **in.TLPHeader_OBFFCode** = 0b1111 : "CPU Active";

in.TLPHeader_Rsvd1[i]: 2nd Reserved field of the TLP Header of the Token TLP Message Packet.

5.2.1.9.14 *USB4 HS Tunneled PCIe TLP PTM Response_ResponseD Packets*

Note: Valid only in case of **in.TLPType[i]** = _USB4_TUN_PCIE_TLP_MSGD. Else – see *USB4 HS Tunneled PCIe TLP Message Reserved Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Message Packet.

in.TLPHeader_PTMMasterTimeHi[i]: Value of the most significant DWORD of PTMMasterTime field of the TLP Header of the Token Packet.

in.TLPHeader_PTMMasterTimeLo[i]: Value of the less significant DWORD of PTMMasterTime field of the TLP Header of the Token Packet.

in.TLPHeader_PropagationDelay[i]: PropagationDelay field of the TLP Header of the Token TLP Message Packet.

5.2.1.9.15 *USB4 HS Tunneled PCIe TLP Vendor Defined Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Message Packet.

in.TLPHeader_VDMSubtype[i]: Subtype of the Token TLP Vendor Defined Message Packet. May be one of the values in table below.

Subtype	Description
_USB4_TUN_PCIE_TLP_MSG_VENDOR_SUBTYPE_LN	Lightweight Notification (LN) Message
_USB4_TUN_PCIE_TLP_MSG_VENDOR_SUBTYPE_HIERARCHY_ID	Hierarchy ID Message
_USB4_TUN_PCIE_TLP_MSG_VENDOR_SUBTYPE_DEVICE_READINESS_STATUS	Device Readiness Status (DRS) Message
_USB4_TUN_PCIE_TLP_MSG_VENDOR_SUBTYPE_FUNCTION_READINESS_STATUS	Function Readiness Status (FRS) Message

in.TLPHeader_VendorID[i]: VendorID field of the TLP Header of the Token TLP Vendor Defined Message Packet.

in.TLPHeader_Rsvd0[i]: 1st Reserved field of the TLP Header of the Token TLP Vendor Defined Message Packet. **in.TLPHeader_Rsvd0[i]** value of this packet repeats **in.Tag[i]** value of TLP Header.

5.2.1.9.16 *USB4 HS Tunneled PCIe TLP Vendor Defined LN Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Vendor Defined Message Packet.

in.DeviceID_BusNumber[i]: DeviceID BusNumber field of the of the Token Packet.

in.DeviceID_DeviceNumber[i]: DeviceID DeviceNumber field of the of the Token Packet.

in.DeviceID_FunctionNumber[i]: DeviceID FunctionNumber field of the of the Token Packet.

in.TLPHeader_VendorRsvd[i]: VendorRsvd field of the of the Token Packet.

in.TLPHeader_Rsvd1[i]: 2nd Reserved field of the TLP Header of the Token Packet.

in.TLPHeader_CachlineAddressHi[i]: Value of the most significant DWORD of CachlineAddress field of the TLP Header of the Token Packet.

in.TLPHeader_CachlineAddressLo[i]: Value of the less significant DWORD of CachlineAddress field of the TLP Header of the Token Packet.

in.TLPHeader_PropagationDelay[i]: PropagationDelay field of the TLP Header of the Token TLP Message Packet.

in.TLPHeader_Rsvd2[i]: 3rd Reserved field of the TLP Header of the Token Packet.

in.TLPHeader_NR[i]: NR field of the TLP Header of the Token Packet.

5.2.1.9.17 *USB4 HS Tunneled PCIe TLP Vendor Defined HierarchyID Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Vendor Defined Message Packet.

in.TLPHeader_HierarchyID[i]: HierarchyID field of the of the Token Packet.

in.TLPHeader_GUIDAuthorityID[i]: GUIDAuthorityID field of the of the Token Packet.
in.TLPHeader_GUIDAuthorityID = 0 : None;
in.TLPHeader_GUIDAuthorityID = 1 : Timestamp;
in.TLPHeader_GUIDAuthorityID = 2 : IEEE_EUI_48;
in.TLPHeader_GUIDAuthorityID = 3 : IEEE_EUI_64;
in.TLPHeader_GUIDAuthorityID = 4 : RFC_4122_UUID;
in.TLPHeader_GUIDAuthorityID = 5 : IPv6Address;
0x06 <= **in.TLPHeader_GUIDAuthorityID** <= 0x7F : Reserved;
0x80 <= **in.TLPHeader_GUIDAuthorityID** : PCI_SIG_Vendor_Specific;

in.TLPHeader_SystemGUIDLen[i]: Count of SystemGUID symbols of the Token Packet.

in.TLPHeader_SystemGUIDList: List of SystemGUID symbols. **in.TLPHeader_SystemGUIDList[i][N]** –Nth SystemGUID symbol of the Token Packet.
(**in.TLPHeader_SystemGUIDList[i][0]**,
in.TLPHeader_SystemGUIDList[i][1],
…,
in.TLPHeader_SystemGUIDList[i][in.TLPHeader_SystemGUIDLen[i]-1]).

5.2.1.9.18 *USB4 HS Tunneled PCIe TLP Vendor Defined DRS Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Vendor Defined Message Packet.

`in.TLPHeader_VendorRsvd[i]:` VendorRsvd field of the of the Token Packet.

`in.TLPHeader_Rsvd1[i]:` Reserved field of the of the Token Packet.

5.2.1.9.19 *USB4 HS Tunneled PCIe TLP Vendor Defined FRS Message Packets*

Note: Packet from this category include all members of Tunneled PCIe TLP Vendor Defined Message Packet.

`in.TLPHeader_VendorRsvd[i]:` VendorRsvd field of the of the Token Packet.

`in.TLPHeader_Rsvd1[i]:` Reserved field of the of the Token Packet.

`in.TLPHeader_FRSReason[i]:` FRSReason field of the of the Token Packet.
- **`in.TLPHeader_FRSReason[i]`** = 0b0001 : DRSMessageReceived;
- **`in.TLPHeader_FRSReason[i]`** = 0b0010 : D3HotD0TransitionCompleted;
- **`in.TLPHeader_FRSReason[i]`** = 0b0011 : FLR Completed;
- **`in.TLPHeader_FRSReason[i]`** = 0b1000 : VF Enabled;
- **`in.TLPHeader_FRSReason[i]`** = 0b1001 : VF Disabled;

5.2.1.9.20 *USB4 HS Tunneled PCIe inner DLLP Packets*

Every field has index of inner packet, which starts with zero.

`in.DLLPType[i]`: Tunneled PCIe DLLP type of the Token Packet. May be one of the values in table below.

DLLP type	Description
_USB4_TUN_PCIE_DLLP_ACK	Ack
_USB4_TUN_PCIE_DLLP_NAK	Nak
_USB4_TUN_PCIE_DLLP_MR_INIT	MRInit
_USB4_TUN_PCIE_DLLP_DATA_LINK_FEATURE	Data_Link_Feature
_USB4_TUN_PCIE_DLLP_PM_ENTER_L1	PM_Enter_L1
_USB4_TUN_PCIE_DLLP_PM_ENTER_L23	PM_Enter_L23
_USB4_TUN_PCIE_DLLP_PM_ACT_STATE_REQUEST_L1	PM_Active_State_Request_L1
_USB4_TUN_PCIE_DLLP_PM_REQUEST_ACK	PM_Request_Ack
_USB4_TUN_PCIE_DLLP_VENDOR	Vendor-specific
_USB4_TUN_PCIE_DLLP_NOP	NOP
_USB4_TUN_PCIE_DLLP_INIT_FC1_P	InitFC1-P

_USB4_TUN_PCIE_DLLP_INIT_FC1_NP	InitFC1-NP
_USB4_TUN_PCIE_DLLP_INIT_FC1_CPL	InitFC1-Cpl
_USB4_TUN_PCIE_DLLP_MR_INIT_FC1	MRInitFC1
_USB4_TUN_PCIE_DLLP_INIT_FC2_P	InitFC2-P
_USB4_TUN_PCIE_DLLP_INIT_FC2_NP	InitFC2-NP
_USB4_TUN_PCIE_DLLP_INIT_FC2_CPL	InitFC2-Cpl
_USB4_TUN_PCIE_DLLP_MR_INIT_FC2	MRInitFC2
_USB4_TUN_PCIE_DLLP_UPDATE_FC_P	UpdateFC-P
_USB4_TUN_PCIE_DLLP_UPDATE_FC_NP	UpdateFC-NP
_USB4_TUN_PCIE_DLLP_UPDATE_FC_CPL	UpdateFC-Cpl
_USB4_TUN_PCIE_DLLP_MR_UPDATE_FC	MRUpdateFC
_USB4_TUN_PCIE_DLLP_INVALID	Invalid DLLP

in.CRC16[i]: Value of CRC16 protecting the payload of the Token Packet.

5.2.1.9.21 *USB4 HS Tunneled PCIe DLLP Ack/Nak Packets*

Note: Packet from this category include all members of Tunneled PCIe DLLP Packet.

in.AckNak_SeqNum[i]: AckNak_SeqNum field of the of the Token Packet.

in.Rsvd[i]: Reserved field of the of the Token Packet.

5.2.1.9.22 *USB4 HS Tunneled PCIe DLLP Data Link Feature Packets*

Note: Packet from this category include all members of Tunneled PCIe DLLP Packet.

in.FeatureAck[i]: FeatureAck field of the of the Token Packet.

in.FeatureSupport[i]: FeatureSupport field of the of the Token Packet.

5.2.1.9.23 *USB4 HS Tunneled PCIe DLLP Power Management Packets*

Note: This category belongs to such DLLP Types:
PM_Enter_L1,
PM_Enter_L23,
PM_Active_State_Request_L1,
PM_Request_Ack.

Note: Packet from this category include all members of Tunneled PCIe DLLP Packet.

in.Rsvd[i]: Reserved field of the of the Token Packet.

5.2.1.9.24 *USB4 HS Tunneled PCIe DLLP Vendor-specific Packets*

Note: Packet from this category include all members of Tunneled PCIe DLLP Packet.

`in.DLLPData[i]:` DLLPData field of the of the Token Packet.

5.2.1.9.25 *USB4 HS Tunneled PCIe DLLP Init/Update Packets*

Note: This category belongs to such DLLP Types:
MRInit,
InitFC1-P,
InitFC1-NP,
InitFC1-Cpl,
MRInitFC1,
InitFC2-P,
InitFC2-NP,
InitFC2-Cpl,
MRInitFC2,
UpdateFC-P,
UpdateFC-NP,
UpdateFC-Cpl,
MRUpdateFC.

Note: Packet from this category include all members of Tunneled PCIe DLLP Packet.

`in.Rsvd[i]:` Reserved field of the of the Token Packet.

`in.HdrScale[i]:` HdrScale field of the of the Token Packet.

`in.HdrFC[i]:` HdrFC field of the of the Token Packet.

`in.DataScale[i]:` DataScale field of the of the Token Packet.

`in.DataFC[i]:` DataFC field of the of the Token Packet.

5.2.1.10 USB4 HS Tunneled USB3 Packet-specific Set of Members

Note: Valid for USB4 HS Tunneled USB3 packets only, undefined for other events.
Note: To get access to given set of members use function SetUsb4TunProtocol().

Note: Tunneled USB3 Set of Members include all members of USB4 HS Tunneled Packet.

in.TunUSB3GenType: Tunneled USB3 Gen type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_TUN_USB3_GEN_X	Gen X
_USB4_TUN_USB3_GEN_T	Gen T

in.IsGenX: Boolean value that indicates whether or not Gen type of the Tunneled USB3 Packet is Gen X.

in.IsGenT: Boolean value that indicates whether or not Gen type of the Tunneled USB3 Packet is Gen T.

in.TunUSB3PacketType: Tunneled USB3 packet type of the Token Packet. May be one of the values in table below.

Packet type	Description
_USB4_TUN_USB3_LFPS	LFPS Packet
_USB4_TUN_USB3_OOBM	OOBM
_USB4_TUN_USB3_TS1	TS1
_USB4_TUN_USB3_TS2	TS2
_USB4_TUN_USB3_SDS	SDS
_USB4_TUN_USB3_LINK_COMMAND	Link Command Packet
_USB4_TUN_USB3_LMP	Link Management Packet
_USB4_TUN_USB3_TP	Transaction Packet
_USB4_TUN_USB3_DEFERRED_DPH	Data Packet Header
_USB4_TUN_USB3_ITP	Isochronous Timestamp Packet
_USB4_TUN_USB3_START_DP	Start Data Packet Segment
_USB4_TUN_USB3_MIDDLE_DP	Middle Data Packet Segment
_USB4_TUN_USB3_END_DP	End Data Packet Segment
_USB4_TUN_USB3_NDP	Nullified Data Packet

5.2.1.10.1 *USB4 HS Tunneled USB3 LFPS Packet Members*

Note: USB4 HS Tunneled USB3 LFPS Packet Members include all members of Tunneled USB3 Packet.

in.Type: Tunneled USB3 LFPS type of the Token Packet. May be one of the values in table below.

LFPS type	Description

_USB4_TUN_USB3_LFPS_POLLING	Polling LFPS
_USB4_TUN_USB3_LFPS_EXIT_U2_LB	Exit U2 LB
_USB4_TUN_USB3_LFPS_WAKE_U3	Wake U3
_USB4_TUN_USB3_LFPS_RESET	Reset LFPS
_USB4_TUN_USB3_LFPS_LBPM_LFPS1	LBPM LFPS1
_USB4_TUN_USB3_LFPS_STOP	Stop LFPS

in.LBPM: LBPM field of the Token Packet.

in.RxTermEnable: RxTermEnable field of the Token Packet.

in.SCD1: SCD1 field of the Token Packet.

in.SCD2: SCD2 field of the Token Packet.

in.U2Exit: U2Exit field of the Token Packet.

in.U3Wakeup: SCD2 field of the Token Packet.

in.WarmReset: WarmReset field of the Token Packet.

in.LBPMEnable: LBPMEnable field of the Token Packet.

in.CRC8: Value of CRC8 protecting the payload of the Token Packet.

Note: Token Packet has **in.Duration** member, common member of USB Packet-specific Set of Members.

5.2.1.10.2 *USB4 HS Tunneled USB3 OOBM Packet Members*

Note: USB4 HS Tunneled USB3 OOBM Packet Members include all members of Tunneled USB3 Packet.

in.LinkError: Link Error field of the Token Packet.

in.PortDisabled: Port Disabled field of the Token Packet.

in.PortResetDone: Port Reset Done field of the Token Packet.

in.U3Resume: U3 Resume field of the Token Packet.

in.PortReset: Port Reset field of the Token Packet.

in.LinkReady: Link Ready field of the Token Packet.

in.CRC: Value of CRC protecting the payload of the Token Packet.

5.2.1.10.3 *USB4 HS Tunneled USB3 TS1/TS2 Ordered Set Members*

Note: USB4 HS Tunneled USB3 TS1/TS2 Members include all members of Tunneled USB3 Packet.

in.LnkFunctionality: Value of the whole Link Functionality (Link Configuration) byte in the training sequence

in.Reset: Value of the Reset bit in the Link Configuration byte.

in.Loopback: Value of the Loopback asserted/deasserted bit in the Link Configuration byte.

in.ScrDisable: Value of the Disable Scrambling bit in the Link Configuration byte.

in.Rsvd0: Rsvd0 field of the of the Token Packet.

in.Rsvd: Rsvd field of the of the Token Packet.

in.CRC8: Value of CRC8 protecting the payload of the Token Packet.

Note: USB4 HS Tunneled USB3 TS1/TS2 has **no** member **in.Count** , since for USB4 tunneled USB3 TS1/TS2 it is always 1.

5.2.1.10.4 *USB4 HS Tunneled USB3 SDS Ordered Set Members*

Note: USB4 HS Tunneled USB3 SDS Members include all members of Tunneled USB3 Packet.

in.CRC8: Value of CRC8 protecting the payload of the Token Packet.

5.2.1.10.5 *USB4 HS Tunneled USB3 Link Command Packet Members*

Note: USB4 HS Tunneled USB3 Link Command Packet Members include all members of Tunneled USB3 Packet.

Note: The following members are actual when the Tunneled USB3 Packet has Gen X Gen Type (**in.TunUSB3GenType** is equal to _USB4_TUN_USB3_GEN_X).

in.LnkCmd: Index of the Link Command type (constants are defined in **VS_constants.inc**).

in.LinkCmdInfo: Value of the Link Command Information field for the Link Command Word.

in.CRC5: Value of CRC5 for the Link Command Word.

in.LinkCmdInfo_2: Value of the Link Command Information field for the replica of the Link Command Word.

in.CRC5_2: Value of CRC5 for the replica of the Link Command Word.

Note: The following members are actual when the Tunneled USB3 Packet has Gen T Gen Type (**in.TunUSB3GenType** is equal to _USB4_TUN_USB3_GEN_T).

in.LnkCmdCount: Count of Link Commands that are encapsulated in the Token Packet.

in.LnkCmd_i: Index of the Link Command type (constants are defined in **VS_constants.inc**). Index i in the prefix may take value from 0 to **in.LnkCmdCount** – 1.

in.LinkCmdInfo_i: Value of the Link Command Information field for the Link Command Word. Index i in the prefix may take value from 0 to **in.LnkCmdCount** – 1.

in.CRC5_i: Value of CRC5 for the Link Command Word. Index i in the prefix may take value from 0 to **in.LnkCmdCount** – 1.

5.2.1.10.6 *USB4 HS Tunneled USB3 Link Management Packet Members*

Note: USB4 HS Tunneled USB3 Link Management Packet Members include all members of Tunneled USB3 Packet.

in.SubType: SubType value for LMP. Use constant definitions in **VS_constants.inc.**

5.2.1.10.7 *USB4 HS Tunneled USB3 Transaction Packet Members*

Note: USB4 HS Tunneled USB3 TP Packet Members include all members of Tunneled USB3 Packet.

in.SubType: SubType value for Transaction Packet. Use constant definitions in **VS_constants.inc.**

in.Addr: Address of the device that receives this Transaction Packet

in.Endp: Endpoint number on the device that receives this Transaction Packet

in.Dir: Direction for the specified Endpoint

in.StreamID: Value of the StreamID field

5.2.1.10.8 *USB4 HS Tunneled USB3 Deferred Data Packet Header Members*

Note: USB4 HS Tunneled USB3 Deferred DPH Members include all members of Tunneled USB3 Packet.

in.Addr: Address of the device which is the recipient of this Data Packet

in.Endp: Endpoint number on the device which is the recipient of this Data Packet

in.Dir: Direction for the specified Endpoint

in.StreamID: Value of the StreamID field

in.SeqN: Value of the Sequence Number for this Data Packett

in.DataLength: Length of the payload for this Data Packet

5.2.1.10.9 *USB4 HS Tunneled USB3 Isochronous Timestamp Packet Members*

Note: USB4 HS Tunneled USB3 ITP Members include all members of Tunneled USB3 Packet.

in.BusICounter: Bus Interval Counter value for ITP, in 1/8 of a millisecond

in.Delta: Time Delta value for ITP

in.BusIAdjCtrl: Bus Interval Adjustment Control value for ITP, specifying the device in control of bus interval adjustment

5.2.1.10.10 *USB4 HS Tunneled USB3 Data Packet Segment Members*

Note: USB4 HS Tunneled USB3 DP Segment Members include all members of Tunneled USB3 Packet.

in.DataLen: Length of Data field in bytes of the Token Packet.

in.Data: Bit source of Data field Of the Token Packet. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

5.2.1.10.11 *USB4 HS Tunneled USB3 Nullified Data Packet Members*

Note: USB4 HS Tunneled USB3 Nullified DP Members include all members of Tunneled USB3 Packet.

in.DataLen: Length of Data field in bytes of the Token Packet.

in.Data: Bit source of Data field Of the Token Packet. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

5.2.1.11 USB4 LLSM State-specific Set of Members

Note: Valid for USB4 LLSM State only, undefined for other events.

in.LaneNumber: Value of the Lane Number of the Trace Event. Value **in.LaneNumber** = 0 means Lane 0, **in.LaneNumber** = 1 means Lane 1.

in.LLSMState: LLSM State into which the transition occured. May be one of the values in table below.

LLSM State	Description

_USB4_LLSM_STATE_UNKNOWN	Unknown
_USB4_LLSM_STATE_TRAINING	Training
_USB4_LLSM_STATE_CLD	CLd
_USB4_LLSM_STATE_CL0	CL0
_USB4_LLSM_STATE_CL0S	CL0s
_USB4_LLSM_STATE_CL1	CL1
_USB4_LLSM_STATE_CL2	CL2
_USB4_LLSM_STATE_LANE_BONDING	Lane Bonding
_USB4_LLSM_STATE_DISABLED	Disabled

5.2.2 Transaction-specific Set of Members

in.TraToken: Token PID for the transaction. Possible values are:
const PID_OUT = 0x87;
const PID_IN = 0x96;
const PID_SETUP = 0xB4;
const PID_SPLIT = 0x1E;
const PID_PING = 0x2D;
const PID_EXT = 0x0F;

in.Addr: Device Address value for this Usb transaction.

in.Endp: Device Endpoint Number value for this Usb transaction.

in.TraCompletionFlag: Handshake PID for the transaction (if any). Possible values are:
const PID_ACK = 0x4B;
const PID_NAK = 0x5A;
const PID_STALL = 0x78;
const PID_NYET = 0x69;
const PID_ERR = 0x3C;
0xFFFFFFFF – in case no handshake was sent for the transaction.

in.Complete: Flag, signaling if the transaction was complete.

in.XferType: Type of the USB Transfer this transaction belongs to. See Transfer Types for possible values.

in.CtrlDirection: For a Setup transaction represents the direction of the corresponding Control Transfer.
0 = Host-to-device
1 = Device-to-host

in.FirstInXfer: This flag is set when the transaction is first in the transfer it belongs to.

in.LastInXfer: This flag is set when the transaction is last in the transfer it belongs to.

in.Errors: Error code for the transaction, if any error happened.

If the transaction is a Setup transaction, the following input context members provide values of the corresponding fields of the USB Device Request:

in.ReqType
in.ReqRecipient
in.bRequest
in.wIndex
in.wValue
in.wLength

in.Split: Signals that the transaction is a start-split or complete-split.

in.HubAddr: Value of the Hub Address field, the USB device address of the hub supporting the specified full-/low-speed device for this full-/low-speed transaction.

in.SC: Value of the Start/Complete bit for the split token.

in.Port: Value of the Port field – the port number of the target hub for which this full-/low-speed transaction is destined.

in.S: Value of the Speed bit for the split token.

in.E: Value of the End bit for the split token.

in.ET: Value of the Endpoint Type field for the split token.

in.OTGHostId: For USB On-The-Go transactions, indicates if the A-device is acting as the host (value of 0) or the B-device is acting as the host (value of 1).

in.OTGErrorCode: For USB On-The-Go transactions, indicates error code if non-zero.

in.OTGErrorCodeOffset: For USB On-The-Go transactions with errors, indicates the offset from the first packet in the transaction to the packet with error.

in.PayloadLength: Length (in bytes) of the packet payload. If zero, the transaction does not have any payload.

in.Payload: Bit source of the transaction data payload. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

5.2.3 Split Transaction-specific Set of Members

All the input context members that are defined for USB2 transactions are also applicable to Split Transactions. In addition to those, Split Transactions define the following members:

`in.TraHalted:` Indicates that this split transaction was halted.

5.2.4 Transfer-specific Set of Members

in.XferType: Type of transfer. Repeats the trace event value for the transfer.

in.Addr: Device Address value for this USB transfer.

in.Endp: Device Endpoint Number value for this USB transfer.

in.Completed: Flag signaling if the transfer was completed.

in.Halted: Flag signaling if the transfer was halted (usually that means STALL handshake sent).

in.CtrlDirection: For a Control Transfer, represents the direction:
0 = Host-to-device
1 = Device-to-host

If the transfer is a Control Transfer, the following input context members provide values of the corresponding fields of the USB Device Request:
in.ReqType
in.ReqRecipient
in.bRequest
in.wIndex
in.wValue
in.wLength

in.ClassicOnHigh: When set, indicates that this transfer was a classic speed transfer that was executed and captured on a high speed branch (upstream from a high speed hub).

in.OTGHostId: For USB On-The-Go transfers, indicates if the A-device is acting as the host (value of 0) or the B-device is acting as the host (value of 1).

in.PayloadLength: Length (in bytes) of the transfer payload. If zero, transfer does not have any payload. Care should be taken not to request payload for the transfers that moved large amounts of data.

in.Payload: Bit source of the transfer payload. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

Note: Refer to the section Script decoded fields retrieving functions for information about getting the script decoded values at the Transfer level.

5.2.5 SCSI Operation-specific Set of Members

SCSI operations define the following members:

`in.OperCode`: SCSI operation code.

`in.Addr`: Device address value for this SCSI operation.

`in.Tag`: SCSI operation tag.

`in.Status`: SCSI operation status.

Metric parameters which are experessed as following members:

`in.XferCount`: The total number of transfers that compose this SCSI operation.

`in.ResponseTime`: Time it took to transmit this SCSI operation on the USB link(s) (VSE Time object).

`in.LatencyTime`: Time measured from the transmission of the SCSI command to the first data transmitted for this SCSI IO operation (VSE Time object).

`in. DataToStatusTime`: Time between the end of data transmission for this SCSI operation and the status transfer (VSE Time object).

`in.Throughput`: payload divided by response time, expressed in **kilobytes** per second.

`in.PldBytes`: Number of payload bytes this SCSI operation transferred.

5.2.6 PHY Transaction-specific Set of Members

PHY Transactions define the following members:

in.Type: PHY Transaction Type. Possible values are:
const _SCD1 = 0x01;
const _SCD2 = 0x02;
const _PHY_CAP_5 = 0x03;
const _PHY_CAP_10 = 0x04;
const _PHY_READY = 0x05;

5.2.7 PD Packet-specific Set of Members

Note 1: All field values can be retrieved via "Input Context" or "Packet And Script Decoded Fields Retrieving Functions". The following list are exceptions to this rule. Some of these members are additional information to decode packets and are not necessary packet fields.

IMPORTANT NOTE:
AS A GENERAL RULE, ADD "OBJx_" PREFIX TO ALL POWER DELIVERY DATA OBJECT FIELD NAMES TO RETRIEVE THEIR VALUE IN A GIVEN PACKET.
x IS THE DATA OBJECT NUMBER WHICH THE FIELD BELONGS TO.
THERE ARE SOME EXCEPTION FIELDS THAT ARE INDICATED BY "DO NOT ADD PREFIX" SIGN.

Here is how to use decoder and context methods to retrieve a field's value:

Method 1:

```
field_prefix =  "OBJ1_";
field_name = field_prefix + "Rsvd_1";
report_str = GetDecodedPktField (field_name);
```

Method 2:

```
field_name = "in." + field_prefix + "Rsvd_1";
report_str = FormatEx( FormatString_IntValue, field_name, in.OBJ1_Rsvd_1 );
```

5.2.7.1 Information Available Only by Context

The following information can just be accessed using input contexts and cannot be retrieved using Packet and Script Decoded Fields Retrieving Functions:

in.PreambleLength: Indicates the Preamble length in bits.

in.PacketType: The 4-bit Message Type field indicating the type of Message being sent.

values for Power Delivery Control, Data, Ordered Set Message Types:

_PD_CONTROL_RSV0 = 0,
_PD_CONTROL_GOOD_CRC = 1,
_PD_CONTROL_GOTO_MIN = 2,

```
_PD_CONTROL_ACCEPT                      = 3,
_PD_CONTROL_REJECT                      = 4,
_PD_CONTROL_PING                        = 5,
_PD_CONTROL_PS_RDY                      = 6,
_PD_CONTROL_GET_SRC_CAP                 = 7,
_PD_CONTROL_GET_SINK_CAP                = 8,
_PD_CONTROL_DR_SWAP                     = 9,
_PD_CONTROL_PR_SWAP                     = 10,
_PD_CONTROL_VCONN_SWAP                  = 11,
_PD_CONTROL_WAIT                        = 12,
_PD_CONTROL_SOFT_RESET                  = 13,
_PD_CONTROL_DATA_RESET                  = 14,
_PD_CONTROL_DATA_RESET_COMPLETE         = 15,
_PD_CONTROL_NOT_SUPPORTED               = 16,
_PD_CONTROL_GET_SRC_CAP_EXTENDED        = 17,
_PD_CONTROL_GET_STATUS                  = 18,
_PD_CONTROL_FR_SWAP                     = 19,
_PD_CONTROL_GET_PPS_STATUS              = 20,
_PD_CONTROL_GET_COUNTRY_CODE            = 21,
_PD_CONTROL_GET_SNK_CAP_EXTENDED        = 22,
_PD_CONTROL_GET_SOURCE_INFO             = 23,
_PD_CONTROL_GET_REVISION                = 24,

_PD_DATA_SRC_CAP                        = 33,
_PD_DATA_REQ                            = 34,
_PD_DATA_BIST                           = 35,
_PD_DATA_SNK_CAP                        = 36,
_PD_DATA_BATTERY_STATUS                 = 37,
_PD_DATA_ALERT                          = 38,
_PD_DATA_GET_COUNTRY_INFO               = 39,
_PD_DATA_ENTER_USB                      = 40,
_PD_DATA_EPR_REQ                        = 41,
_PD_DATA_EPR_MODE                       = 42,
_PD_DATA_SRC_INFO                       = 43,
_PD_DATA_REVISION                       = 44,
_PD_DATA_VENDOR_DEF                     = 47,

_PD_EXTENDED_SRC_CAP_EXTENDED           = 65,
_PD_EXTENDED_STATUS                     = 66,
_PD_EXTENDED_GET_BATTERY_CAP            = 67,
_PD_EXTENDED_GET_BATTERY_STATUS         = 68,
_PD_EXTENDED_BATTERY_CAP                = 69,
_PD_EXTENDED_GET_MANUFACTURER_INFO      = 70,
_PD_EXTENDED_MANUFACTURER_INFO          = 71,
_PD_EXTENDED_SECURITY_REQ               = 72,
_PD_EXTENDED_SECURITY_RESP              = 73,
_PD_EXTENDED_FW_UPDATE_REQ              = 74,
_PD_EXTENDED_FW_UPDATE_RESP             = 75,
_PD_EXTENDED_PPS_STATUS                 = 76,
_PD_EXTENDED_COUNTRY_INFO               = 77,
_PD_EXTENDED_COUNTRY_CODE               = 78,
_PD_EXTENDED_SNK_CAP_EXTENDED           = 79,
```

_PD_EXTENDED_EPR_SRC_CAPABILITIES……..= 81,
_PD_EXTENDED_EPR_SNK_CAPABILITIES = 82,
_PD_EXTENDED_VENDOR_DEFINED_EXT = 83

_PD_EXTENDED_CONTROL_EPR_GET_SRC_CAP = 97,
_PD_EXTENDED_CONTROL_EPR_GET_SNK_CAP = 98,
_PD_EXTENDED_CONTROL_EPR_KEEP_ALIVE = 99,
_PD_EXTENDED_CONTROL_EPR_KEEP_ALIVE_ACK = 100,

_PD_MESSAGE_TYPE_END___ = 249

_PD_MSG_EVENT_FAST_POWER_ROLE_SWAP= 0xfa,
_PD_MSG_HARD_RESET = 0xfb,
_PD_MSG_CABLE_RESET = 0xfc,
_PD_OTHER_MSG_BIST_TEST_STREAM = 0xfd,
_PD_MSG_UNKNOWN = 0xfe,

in.SenderPort: The port sent this packet according to data provided by Bus Engine (front end acquizition hardware).

It can be one of the following values:

_PD_PORT_UNKNOWN = 0
_PD_PORT_LEFT = 1
_PD_PORT_RIGHT = 2
_PD_PORT_CABLE = 3

in.SOP_Type: Value of SOP Type for Start of Packet

It can be one of the following values:

_PD_FRAMING_TYPE_NO_FRAMING = 0,
_PD_FRAMING_TYPE_SOP = 1,
_PD_FRAMING_TYPE_SOP_PRIM = 2,
_PD_FRAMING_TYPE_SOP_DOUBLE_PRIM = 3,
_PD_FRAMING_TYPE_SOP_PRIM_DBG = 4,
_PD_FRAMING_TYPE_SOP_DBL_PRIM_DBG = 5,
_PD_FRAMING_TYPE_HARD_RESET = 6,
_PD_FRAMING_TYPE_CABLE_RESET = 7,

in.DataObjectCount: The 3-bit Number of Data Objects field shall indicate the number of 32-bit Data Objects that follow the Message Header. When this field is zero the Message is either a Control Message or an unchunked extended message, otherwise the message is either a Data Message or a chunked extended message.

in. OBJX_CapType: Capability type of the data object X which can be either _PD_DATA_SNK_CAP or _PD_DATA_SRC_CAP for Source_Capabilities or Sink_Capabilities capability messages

in.OBJ1_MinOprCurPow: The Minimum Operating Current/Power field in the Request Message shall be set to the lowest current the Sink requires to maintain operation.
The Values returned are in 10mA and 250mW for Current and Power members respectively.

in.OBJ1_MaxOprCurPow: The Maximum Operating Current/Power field in the Request Message shall be set to the highest current the Sink will ever require .
The Values returned are in 10mA and 250mW for Current and Power members respectively.

in.OprCurPow: The Operating Current field in the Request Data Object shall be set to the actual amount of current the Sink needs to operate at a given time.
he Values returned are in 10mA and 250mW for Current and Power members respectively.

in.BISTFunction: Type of the BIST message function

It can be one of the following values:

5: BIST Carrier Mode 2
8: BIST Test Data

Following constants are defined:
_PD_BIST_REQ_TYPE_CARRIER_MODE
_PD_BIST_REQ_TYPE_TEST_DATA

in.IsStructuredVDM: A value of 1 indicates current Vendor Defined Message is a Structured VDM, 0 means an Unstructured VDM.

in.VDMCommand: The VDM Header for a Structured VDM Message defines Commands used to retrieve a list of SVIDs the device supports, to discover the Modes associated with each SVID, and to enter/exit the Modes.

The Commands include:
_PD_VDM_CMD_DISCOVER_IDENTITY: Discover Identity
_PD_VDM_CMD_DISCOVER_SVIDS: Discover SVIDs

_PD_VDM_CMD_DISCOVER_MODES: Discover Modes
_PD_VDM_CMD_ENTER_MODE: Enter Mode
_PD_VDM_CMD_EXIT_MODE: Exit Mode
_PD_VDM_CMD_ATTENTION: Attention

_PD_VDM_CMD_SVID_SPECIFIC_FIRST___
_PD_VDM_CMD_SVID_SPECIFIC_LAST___
_PD_VDM_CMD_SVID_SPECIFIC_1
_PD_VDM_CMD_SVID_SPECIFIC_15

Additional Command space is also reserved for Standard and Vendor use and for future extensions.

in.VDMCommandType: This field shall be used to indicate the type of Command request/response being sent.

Command Type	Meaning
_PD_VDM_CMD_TYPE_INIT	An Initiator shall set the field to “Initiator” to indicate that this is a Command request from an Initiator
_PD_VDM_CMD_TYPE_RESP_ACK	“Responder ACK” is the normal return and shall be sent to indicate that the Command request was received and

	handled normally
_PD_VDM_CMD_TYPE_RESP_NAK	“Responder NAK” shall be returned when the Command request has an invalid parameter (e.g. invalid SVID or Mode) or cannot not be acted upon at this time (e.g. a Mode which has a dependency on another Mode).
_PD_VDM_CMD_TYPE_RESP_BUSY	“Responder BUSY” shall be sent in the response to a VDM when the Responder is unable to respond to the Command request immediately, but the Command request may be retried.

in.VDMVendorID: Unique 16-bit unsigned integer. Assigned by the USB-IF to the Vendor.

in.UndefinedObjectsCount: When data objects are decoded, some data may be recognized as undefined depending on the packet type being decoded, this member gives the number of these undefined data objects.

in.BufferLen: Size of power delivery message in bytes, this includes header and data objects.

5.2.7.2 Message Header Members

in.Msg_Type (Given in PacketType format in Context too)**:** The Message Type field shall indicate the type of Message being sent.

in.Rsvd_1: Reserved

in.DR: The 1-bit Port Data Role field shall indicate the current data role of the Port:
0b UFP
1b DFP

in.Spec_Rev: The 2-bit Specification Revision field shall indicate the revision of the Power Delivery Specification supported by the
Device.
00b – Revision 1.0
01b – Revision 2.0
10b – Revision 3.0
11b – Reserved, shall not be used
Constants available:
_PD_SPEC_REV_1
_PD_SPEC_REV_2
_PD_SPEC_REV_3
_PD_SPEC_REV_UNDEF

in.PR: The 1-bit Port Data Role field shall indicate the current data role of the Port:
0b UFP
1b DFP

in.Cable_Plug: The 1-bit Cable Plug field shall indicate whether this Message originated from a Cable Plug:
0b Message originated from a DFP or UFP
1b Message originated from a Cable Plug
The Cable Plug field shall only apply to SOP’ and SOP’’ Packet types.

in.Msg_ID: The 3-bit MessageID field is the value generated by a rolling counter maintained by the originator of the Message.

in.Obj_Cnt: The 3-bit Number of Data Objects field shall indicate the number of 32-bit Data Objects that follow the Message Header. When this field is zero the Message is a Control Message and when it is non-zero, the Message is a Data Message.
The Number of Data Objects field shall apply to all SOP* Packet types.

Rev2: **in.Rsvd_2:** Reserved
Rev3: **in.Extended:** The 1-bit flag indicates if this is an Extended message or not.

5.2.7.3 Capability Message

Note 1: Power Data Objects (PDO) are identified by the Message Header's Type field. They are used to form Source_Capabilities Messages and Sink_Capabilities Messages.
Note 2: If Message Header's Type field does not contain a valid value the fields will be decoded as an undefined.

5.2.7.3.1 *Source Capabilities Message*

Note 1: There are 4 types of Source Capabilities Message Power Data Objects (PDO) in this category:
Fixed Power Supply
Battery Power Supply
Variable Power Supply
Programmable Power Supply

This can be decided using following member:
in.Supply_Type:

_PD_CAP_PWR_FIXED = 0,
_PD_CAP_PWR_BATTERY = 1,
_PD_CAP_PWR_VARIABLE = 2,

Rev 2.0 : _PD_CAP_PWR_RSVD = 3,
Rev 3.0 : _PD_CAP_PWR_APDO = 3,

5.2.7.3.1.1 Source Fixed Power Supply Data Object Field Names

in.Max_Cur: Maximum Current in 10mA units

in.Voltage: Voltage in 50mV units

in.Peak_Curr:

00 Peak current equals IOC (default)
01 Overload Capabilities:
1. Peak current equals 150% IOC for 1ms @ 5% duty cycle (low current equals 97% IOC for 19ms)
2. Peak current equals 125% IOC for 2ms @ 10% duty cycle (low current equals 97% IOC for 18ms)

3. Peak current equals 110% IOC for 10ms @ 50% duty cycle (low current equals 90% IOC for 10ms)

10 Overload Capabilities:

1. Peak current equals 200% IOC for 1ms @ 5% duty cycle (low current equals 95% IOC for 19ms)
2. Peak current equals 150% IOC for 2ms @ 10% duty cycle (low current equals 94% IOC for 18ms)
3. Peak current equals 125% IOC for 10ms @ 50% duty cycle (low current equals 75% IOC for 10ms)

11 Overload Capabilities:

1. Peak current equals 200% IOC for 1ms @ 5% duty cycle (low current equals 95% IOC for 19ms)
2. Peak current equals 175% IOC for 2ms @ 10% duty cycle (low current equals 92% IOC for 18ms)
3. Peak current equals 150% IOC for 10ms @ 50% duty cycle (low current equals 50% IOC for 10ms)

in.Rsvd: Reserved

in.Unchunked_Ext: The 1-bit flag indicates whether the port supports unchunked extended messages or not.

in.DR_Swap_Supp: The Data Role Swap bit shall be set when the Port is Type-C (see [USBType-C 1.0]) and supports the DR_Swap Message. This is a static capability which shall remain fixed for a given device.

in.USB_Com_Cap: The USB Communications Capable bit shall only be set for devices capable of communication over the USB data lines (e.g. D+/- or SS Tx/Rx).

in.Ext_Pow: The Externally Powered bit

in.USB_Suspend: Prior to a Contract or when the USB Communications Capable bit is set to zero, this flag is undefined and Sinks shall follow the rules for suspend as defined in [USB 2.0], [USB 3.1], [USBType-C 1.0] or [BC 1.2]. After a Contract has been negotiated:

If the USB Suspend Supported flag is set, then the Sink shall follow the [USB 2.0] or [USB 3.1] rules for suspend and resume. A PDUSB Peripheral may draw up to pSnkSusp during suspend; a PDUSB Hub may draw up to pHubSusp during suspend (see Section 7.2.4).

If the USB Suspend Supported flag is cleared, then the Sink shall not apply the [USB 2.0] or [USB 3.1] rules for suspend and may continue to draw the negotiated power. Note that when USB is suspended, the USB device state is also suspended.
Sinks may indicate to the Source that they would prefer to have the USB Suspend Supported flag cleared by setting the No USB Suspend flag in a Request Message

in.Dual_Role: The Dual-Role Power bit shall be set when the Port is Dual-Role Power capable i.e. supports the PR_Swap Message. This is a static capability which shall remain fixed for a given device.

in.Supply_Type: Fixed Supply (00b) PDO

5.2.7.3.1.2 Source Battery Power Supply Data Object Field Names

in.Max_Allowed_Pwr: Maximum Allowable Power in 250mW units

in.Min_Volt: Minimum Voltage in 50mV units

in.Max_Volt: Maximum Voltage in 50mV units

in.Supply_Type: Battery

5.2.7.3.1.3 Source Variable Power Supply Data Object Field Names

in.Max_Cur: Maximum Current in 10mA units

in.Min_Volt: Minimum Voltage in 50mV units

in.Max_Volt: Maximum Voltage in 50mV units

in.Supply_Type: Variable Supply (non-battery)

5.2.7.3.1.4 Source Augmented Power Data Object (APDO) Field Names

This can be decided using following member:
in.Sub_Type:

_PD_CAP_APDO_PPS = 0,
_PD_CAP_APDO_EPR_AVS = 1,
_PD_CAP_APDO_SPR_AVS = 2,

_PD_CAP_APDO_RSVD2 = 3,

5.2.7.3.1.4.1 SPR Programmable Power Supply (PPS) APDO Field Names

in.Max_Cur: Max current in 50mA

in.Rsvd_1: Reserved

in.Min_Volt: Minimum voltage in 100mV

in.Rsvd_2: Reserved

in.Max_Volt: Maximum voltage in 100mV

in.Rsvd_3: Reserved

in.Power_Ltd: PPS Power Limited

in.Sub_Type: 0x00: SPR-PPS

in.Supply_Type: 0x03: APDO

5.2.7.3.1.4.2 EPR Adjustable Voltage Supply (AVS) APDO Field Names

in.PDP: Power rating in 1W

in.Min_Volt: Minimum voltage in 100mV

in.Rsvd_1: Reserved

in.Max_Volt: Maximum voltage in 100mV

in.Peak_Curr: Peak Current

in.Sub_Type: 0x01: EPR-AVS

in.Supply_Type: 0x03: APDO

5.2.7.3.1.4.3 SPR Adjustable Voltage Supply (AVS) APDO Field Names

in.Max_Cur_20v: Maximum Current - Range 15-20v

in.Max_Cur_15v: Maximum Current - Range 9-15v

in.Rsvd_1: Reserved

in.Peak_Curr: Peak Current

in.Sub_Type: 0x02: SPR-AVS

in.Supply_Type: 0x03: APDO

5.2.7.3.2 *Sink Capabilities Message*

Note 1: There are 4 types of Sink Capabilities Message Power Data Objects (PDO) in this category:
Fixed Power Supply
Battery Power Supply
Variable Power Supply
Programmable Power Supply

This can be decided using following member:
in.Supply_Type:

_PD_CAP_PWR_FIXED = 0,
_PD_CAP_PWR_BATTERY = 1,
_PD_CAP_PWR_VARIABLE = 2,

Rev 2.0 : _PD_CAP_PWR_RSVD = 3,

Rev 3.0 : _PD_CAP_PWR_APDO = 3,

5.2.7.3.2.1 Sink Fixed Power Supply Data Object Field Names

in.Opr_Cur: Operational Current in 10mA units

in.Voltage: Voltage in 50mV units

in.Rsvd: Reserved

in.DR_Swap_Supp: The Data Role Swap bit shall be set when the Port is a Type-C DRP (see [USBType-C 1.0]) and supports the DR_Swap Message. This is a static capability which shall remain fixed for a given device.

in.USB_Com_Cap: The USB Communications Capable bit shall only be set for devices capable of communication over the USB data lines (e.g. D+/- or SS Tx/Rx).

in.Ext_Pow: The Externally Powered bit

in.Higher_Cap: In the case that the Sink needs more than vSafe5V (e.g. 12V) to provide full functionality, then the Higher Capability bit shall be set.

in.Dual_Role: The Dual-Role bit shall be set when the Port is Dual-Role Power capable i.e. supports the PR_Swap Message. This is a static capability which shall remain fixed for a given device.

in.Supply_Type: Fixed supply

5.2.7.3.2.2 Sink Battery Power Supply Data Object Field Names

in.Opr_Pwr: Operational Power in 250mW units

in.Min_Volt: Minimum Voltage in 50mV units

in.Max_Volt: Maximum Voltage in 50mV units

in.Supply_Type: Battery

5.2.7.3.2.3 Sink Variable Power Supply Data Object Field Names

in.Opr_Cur: Operational Current in 10mA units

in.Min_Volt: Minimum Voltage in 50mV units

in.Max_Volt: Maximum Voltage in 50mV units

in.Supply_Type: Variable Supply (non-battery)

5.2.7.3.2.4 Sink Programmable Power Supply APDO Field Names

This can be decided using following member:

in.Sub_Type:

_PD_CAP_APDO_PPS = 0,
_PD_CAP_APDO_EPR_AVS = 1,

_PD_CAP_APDO_RSVD1 = 2,
_PD_CAP_APDO_RSVD2 = 3,

5.2.7.3.2.4.1 SPR Programmable Power Supply APDO Field Names

in.Max_Cur: Max current in 50mA

in.Rsvd_1: Reserved

in.Min_Volt: Minimum voltage in 100mV

in.Rsvd_2: Reserved

in.Max_Volt: Maximum voltage in 100mV

in.Rsvd_3: Reserved

in.Sub_Type: 0x00: PPS

in.Supply_Type: 0x03: APDO

5.2.7.3.2.4.2 EPR Programmable Power Supply APDO Field Names

in.PDP: Power rating in 1W

in.Min_Volt: Minimum voltage in 100mV

in.Rsvd_1: Reserved

in.Max_Volt: Maximum voltage in 100mV

in.Rsvd_2: Reserved

in.Sub_Type: 0x01: AVS

in.Supply_Type: 0x03: APDO

5.2.7.3.3 *Undefined Capabilities Message*

in.Object: Undefined capabilities message data

in.Supply_Type: Value of undefined supply type

5.2.7.4 Request Packet Data Field Names

Note 1: Request packet prefix is always "obj1_"

Note: Following 3 field values can only be accessed using "Input Context"

in.MinOprCurPow: Minimum Operating Current 10mA units

in.MaxOprCurPow: Maximum Operating Power in 250mW units

in.OprCurPow: Operating Power in 250mW units

Following field values can be retrieved via "Input Context" or "Packet And Script Decoded Fields Retrieving Functions"

in.Rsvd_1: Reserved - shall be set to zero.

in.Unchunked_Ext: Unchunked Extended Messages Supported

in.No_USB_Suspend: The No USB Suspend

in.USB_Com_cap: The USB Communications Capable

in.Cap_Mismatch: Capability Mismatch

in.OBJ1_Give_Back: The GiveBack flag shall be set to indicate that the Sink will respond to a GotoMin Message by reducing its load to the Minimum Operating Current. It will typically be used by a USB Device while charging its battery because a short interruption of the charge will have minimal impact on the user and will allow the Source to manage its load better.

in.Obj_Pos: The value in the Object Position field shall indicate which object in the Source_Capabilities Message the RDO refers.
The value 1 always indicates the 5V Fixed Supply PDO as it is the first object following the Source_Capabilities Message Header. The number 2 refers to the next PDO and so forth.

in.Rsvd_2: Reserved - shall be set to zero.

Programmable Power Supply RDO Field Names:
in.Opr_Current: Operating Current 50mA units
in.Output_Voltage: Output Voltage in 20mV
in.Rsvd_3: Reserved
in.Rsvd_4: Reserved

5.2.7.5 Undefined Data Objects Field Names

Do not add prefix to these fields.

To retrieve field value using "Packet And Script Decoded Fields Retrieving Functions", field name format should be:
`Object X`

To retrieve field value using "Input Context", field name format should be:
`in.Object_X`

Where X is a number between 1 and **`in.UndefinedObjectsCount`**

5.2.7.6 BIST Message

`in.Function:` Type of the BIST message function

It can be one of the following values:

2: Returned BIST Counters
5: BIST Carrier Mode
8: BIST Test Data
Other values are reserved.

`Depending on BIST message Function, BIST Message Data Objects are decoded as follows:`

5.2.7.6.1 *BIST Returned Counter Packet Data*

`in.Rsvd:` Reserved and shall be set to zero.

`in.Err_Cnt:` When Request Type is Returned BIST Counters, this field shall contain the contents of BISTErrorCounter otherwise it shall be set to zero.
`in.Function:` Type of the BIST message function

Value is: 2: Returned BIST Counters

5.2.7.6.2 *BIST Packet Data*

`in.Rsvd:` Reserved and shall be set to zero.

`in.Function:` Type of the BIST message function

Value is: 2: Returned BIST Counters

It can be one of the following values:

5: BIST Carrier Mode
8: BIST Test Data

5.2.7.6.3 *BIST Test Data*

in.Data: Test Data Frame

5.2.7.7 BIST Test Stream

When in.PacketType is _PD_OTHER_MSG_BIST_TEST_STREAM, the PDO s decoded as a BIST Test Stream

NOTE: THESE FIELDS CAN ONLY BE ACCESSED USING "INPUT CONTEXT"

in.Mode:

Contains following values:
0: bist test type unknown
1: bist test type carrier mode

in.Length
in.CorruptedDataLen

if **in.CorruptedDataLen** > 0 following fields are valid:

in.Corrupted_Bits
in.CorruptedDataLastByteUnusedBits

5.2.7.8 Vendor Defined Message

5.2.7.8.1 *Vendor Defined VDM Header Field Names*

Note: VDM header prefix is always "obj1_"

For structured VDMs:
in.Cmd:

0 = Reserved, shall not be used
1 = Discover Identity
2 = Discover SVIDs
3 = Discover Modes
4 = Enter Mode
5 = Exit Mode
6 = Attention

7-15 = Reserved, shall not be used
16…31 = SVID Specific Commands

in.Rsvd_1: Shall be set to 0 and shall be Ignored

in.Cmd_Type:

00b = Initiator
01b = Responder ACK
10b = Responder NAK
11b = Responder BUSY

in.Obj_Pos:

For the Enter Mode, Exit Mode and Attention Commands:

000b = Reserved and shall not be used.
001b..110b = Index into the list of VDOs to identify the desired Mode VDO
111b = Exit all Active Modes (equivalent of a power on reset). Shall only be used with the Exit Mode Command.

Commands 0..3, 7..15:

000b
001b..111b = Reserved and shall not be used.

SVID Specific Commands (16..31) defined by the SVID.

in.Rsvd_2: For Commands 0..15 shall be set to 0 and shall be Ignored SVID Specific Commands (16..31) defined by the SVID.

in.VDM_Ver: Version Number of the Structured VDM (not this specification Version):

Version 1.0 = 0
Values 1-3 are Reserved and shall not be used

For unstructured VDMs (Accessed just by GetDecodedPktField function):
Data (15 bits)

Common fields:
in.Type: 1 = Structured VDM

in.Vendor_ID: Unique 16 bit unsigned integer, assigned by the USB-IF

5.2.7.8.2 *Vendor Defined Unstructured Data Objects Field Names*

Do not add prefix to these fields.

To retrieve field value using "Packet And Script Decoded Fields Retrieving Functions", field name format should be:
Object X

To retrieve field value using "Input Context", field name format should be:
in.Object_X

Where X is a number between 1 and **in.DataObjectCount**

5.2.7.8.3 *Vendor Defined Discover ID Response Id Header VDO Field Names*

in.Vendor_ID: Unique 16-bit unsigned integer. Assigned by the USB-IF to the Vendor.

in.Rsvd: Reserved

in.Connector_Type: Connector Type

00b – Reserved, for compatibility with legacy systems.
01b – Reserved, Shall Not be used.
10b – USB Type-C Receptacle
11b – USB Type-C Plug

in.DFP_Type: (rev 3.0)

DFP Type Values:
000b – Undefined
001b – PDUSB Hub
010b – PDUSB Host
011b – Power Brick
100b – Alternate Mode Controller (AMC)
101b … 111b – Reserved

Available constants:
```
_PD_VDM_DFP_TYPE_UNDEFINED
_PD_VDM_DFP_TYPE_HUB
_PD_VDM_DFP_TYPE_HOST
_PD_VDM_DFP_TYPE_POWER_BRICK
_PD_VDM_DFP_TYPE_ALTERNATE_MODE_CONTROLLER
```

in.Modal_Opr:

Modal Operation Supported:
Shall be set to one if the product supports Modal Operation.
Shall be set to zero otherwise

in.Prod_Type:

Product Type UFP:

000b – Undefined
001b – Hub
010b – Peripheral
011b – PSD (Rev 3.0 only Rev 2.0 is Reserved)
100b - Reserved

101b - AMA
110b - VPD (Rev 3.0 only Rev 2.0 is Reserved)
111b - Reserved

Product Type Cable Plug:

000b – Undefined
001b..010 – Reserved
011b – Passive Cable
100b - Active Cable
101b..111b - Reserved

Available constants:
```
_PD_VDM_PRODUCT_TYPE_UNDEFINED
_PD_VDM_PRODUCT_TYPE_HUB
_PD_VDM_PRODUCT_TYPE_PERIPHERAL
_PD_VDM_PRODUCT_TYPE_PSD
_PD_VDM_PRODUCT_TYPE_PASSIVE_CABLE
_PD_VDM_PRODUCT_TYPE_ACTIVE_CABLE
_PD_VDM_PRODUCT_TYPE_ALTERNATE_MODE_ADAPTER
_PD_VDM_PRODUCT_TYPE_VPD
```

in.Device_Cap

Data Capable as a USB Device:
Shall be set to one if the product is capable of enumerating as a USB Device.
Shall be set to zero otherwise

in.Host_Cap:

Data Capable as USB Host:
Shall be set to one if the product is capable of enumerating USB Devices.
Shall be set to zero otherwise

5.2.7.8.4 *Vendor Defined Discover ID Response Cert State VDO Field Names*

in.XID: Assigned by USB-IF

5.2.7.8.5 *Vendor Defined Discover ID Response Product VDO Field Names*

in.bcd_Device: 16-bit unsigned integer. USB Product ID

in.Product_ID: 16-bit unsigned integer. bcdDevice

5.2.7.8.6 *Vendor Defined Discover ID Response Cable VDO Field Names (Rev_2.0)*

in.USB_SS: USB Super speed Signaling Support

000b = USB 2.0 only
001b = [USB 3.1] Gen1
010b = [USB 3.1] Gen1 and Gen2
011b.. 111b = Reserved, shall not be used

in.SOPDPrime_Con: SOP" controller present?

1 = SOP" controller present
0 = No SOP" controller present

in.VBUS_Cable: VBUS through cable

0 = No
1 = Yes

in.VBUS_Curr_Cap: VBUS Current Handling Capability

00b = VBUS not through cable
01b = 3A
10b = 5A
11b = Reserved, shall not be used

in.SS_RX2: SSRX2 Directionality Support

0 = Fixed
1 = Configurable

in.SS_RX1: SSRX1 Directionality Support

0 = Fixed
1 = Configurable

in.SS_TX2: SSTX2 Directionality Support

0 = Fixed
1 = Configurable

in.SS_TX1: SSTX1 Directionality Support

0 = Fixed
1 = Configurable

in.Cable_Term: Cable Termination Type

00b = Both ends Passive, VCONN not required
01b = Both ends Passive, VCONN required
10b = One end Active, one end passive, VCONN required
11b = Both ends Active, VCONN required

in.Cable_Latency: Cable Latency 0000b – Reserved, shall not be used

0001b – <10ns (~1m)
0010b – 10ns to 20ns (~2m)
0011b – 20ns to 30ns (~3m)
0100b – 30ns to 40ns (~4m)
0101b – 40ns to 50ns (~5m)
0110b – 50ns to 60ns (~6m)
0111b – 60ns to 70ns (~7m)
1000b –1000ns (~100m)
1001b –2000ns (~200m)
1010b – 3000ns (~300m)
1011b ….1111b Reserved, shall not be used
Includes latency of electronics in Active Cable

in.Type_C_to_Plg: Type-C plug to

Plug/Receptacle
0 = Plug
1 = Receptacle (not valid when B19..18 set to Type-C or Captive)

in.Type_C_to: Type-C plug to TypeA/B/C/Captive

00b = Type-A
01b = Type-B
10b = Type-C
11b = Captive

in.Rsvd: Reserved

Shall be set to zero

in.FW_Ver: Firmware Version

0000b..1111b assigned by the VID owner

in.HW_Ver: HW Version

0000b..1111b assigned by the VID owner

5.2.7.8.7 *Vendor Defined Discover ID Response UFP VDO 1 Field Names (Rev_3.0)*

in.USB_Highest_Speed: USB Highest Speed

000b = USB 2.0 only, no SuperSpeed support
001b = [USB 3.2] Gen1
010b = [USB 3.2] / [USB4] Gen2
011b = [USB4] Gen3
100b…111b = Reserved, shall not be used

in.Alternate_Modes: Alternate Modes

Bit 0 = Supports [TBT3] Alternate Mode
Bit 1 = Supports Alternate Modes that reconfigure the signals on the [USB Type-C 2.0] connector – except for [TBT3]
Bit 2 = Supports Alternate Modes that do not reconfigure the signals on the [USB Type-C 2.0] connector

in.RSVD1: Reserved

in.Device_Capability: Device Capability

Bit 0 = [USB 2.0] Device Capable
Bit 1 = [USB 2.0] Device Capable (Billboard only)
Bit 2 = [USB 3.2] Device Capable
Bit 3 = [USB4] Device Capable

in.RSVD2: Reserved

in.UFP_VDO_Version: UFP VDO Version

000b – Version 1
001b … 111b – Reserved and Shall Not be used

5.2.7.8.8 *Vendor Defined Discover ID Response UFP VDO 2 Field Names (Rev_3.0)*

in.USB3_Max_Power: USB3 Max Power

Power in watts required for full functionality excluding any power required for battery charging or for redistribution in [USB 3.2] operation.

in.USB3_Min_Power: USB3 Min Power

Minimum power in watts required to function in [USB 3.2] operation.

in.RSVD1: Reserved

in.USB4_Max_Power: USB4 Max Power

Power in watts required for full functionality excluding any power required for battery charging or for redistribution in [USB4] operation.

in.USB4_Min_Power: USB4 Min Power

Minimum power in watts required to function in [USB4] operation.

in.RSVD2: Reserved

Note: if UFP VDO version is 1.0, and 1.1, the above fields will be available. If the version is higher, the following field is only accessible:

in.Padding: Pad Object which should be zero value.

5.2.7.8.9 *Vendor Defined Discover ID Response DFP VDO Field Names (Rev_3.0)*

in.Port_Number: Port Number

Unique port number to identify a specific port on a multi-port device.

in.RSVD1: Reserved

in.Host_Capability: Host Capability

Bit 0 = [USB 2.0] Host Capable
Bit 1 = [USB 3.2] Host Capable
Bit 2 = [USB4] Host Capable

in.RSVD2: Reserved

in.DFP_VDO_Version: DFP VDO Version

000b – Version 1
001b … 111b – Reserved and Shall Not be used

5.2.7.8.10 *Vendor Defined Discover ID Response Passive Cable VDO Field Names (Rev_3.0)*

in.USB_Highest_Speed: USB Highest Speed

000b = USB 2.0 only, no SuperSpeed support
001b = [USB 3.2] Gen1
010b = [USB 3.2] / [USB4] Gen2
011b = [USB4] Gen3
100b…111b = Reserved, shall not be used

in.Rsvd_1: Reserved

in.VBUS_Curr_Cap: VBUS Current Handling Capability

00b = VBUS not through cable
01b = 3A
10b = 5A
11b = Reserved, shall not be used

in.Rsvd_2: Reserved

in.Max_VBus: Maximum Cable VBus Voltage

00b – 20V
01b – 30V
10b – 40V
11b – 50V

in.Cable_Term: Cable Termination Type

00b = Both ends Passive, VCONN not required
01b = Both ends Passive, VCONN required
10b = One end Active, one end passive, VCONN required
11b = Both ends Active, VCONN required

in.Cable_Latency: Cable Latency 0000b – Reserved, shall not be used

0001b – <10ns (~1m)
0010b – 10ns to 20ns (~2m)
0011b – 20ns to 30ns (~3m)
0100b – 30ns to 40ns (~4m)
0101b – 40ns to 50ns (~5m)
0110b – 50ns to 60ns (~6m)
0111b – 60ns to 70ns (~7m)
1000b –1000ns (~100m)
1001b –2000ns (~200m)
1010b – 3000ns (~300m)
1011b ….1111b Reserved, shall not be used
Includes latency of electronics in Active Cable

in.Rsvd_3: Reserved

in.Type_C_to: Type-C plug to Type-C / Captive

00b = Reserved
01b = Reserved
10b = Type-C
11b = Captive

in.Rsvd_4: Reserved

in.VDO_Ver: VDO Version

000b – Version 1
001b … 111b – Reserved

in.FW_Ver: Firmware Version

0000b..1111b assigned by the VID owner

in.HW_Ver: HW Version

0000b..1111b assigned by the VID owner

5.2.7.8.11 *Vendor Defined Discover ID Response Active Cable VDO 1 Field Names (Rev_3.0)*

in.USB_Highest_Speed: USB Highest Speed

000b = USB 2.0 only, no SuperSpeed support
001b = [USB 3.2] Gen1
010b = [USB 3.2] / [USB4] Gen2
011b = [USB4] Gen3
100b = [USB4] Gen4
101b…111b = Reserved, shall not be used

in.SOPDPrime_Con: SOP" controller present?

1 = SOP" controller present
0 = No SOP" controller present

in.VBUS_Cable: VBUS through cable

0 = No
1 = Yes

in.VBUS_Curr_Cap:
When V_{BUS} Through Cable is "No", this field ***Shall*** be ***Ignored***.
When V_{BUS} Though Cable is "Yes":
00b = USB Type-C Default Current
01b = 3A
10b = 5A
11b = ***Reserved***, ***Shall Not*** be used.

in.SBU_Type:
When SBU Supported = 1 this bit ***Shall*** be ***Ignored***
When SBU Supported = 0:
0 = SBU is passive
1 = SBU is active

in.SBU_Supported:
0 = SBUs connections supported
1 = SBU connections are not supported

in.MaxVBus: Maximum Cable VBus Voltage

00b – 20V
01b – 30V
10b – 40V
11b – 50V

in.Cable_Term: Cable Termination Type

00b = Both ends Passive, VCONN not required
01b = Both ends Passive, VCONN required
10b = One end Active, one end passive, VCONN required
11b = Both ends Active, VCONN required

in.Cable_Latency: Cable Latency 0000b – Reserved, shall not be used

0001b – <10ns (~1m)
0010b – 10ns to 20ns (~2m)
0011b – 20ns to 30ns (~3m)
0100b – 30ns to 40ns (~4m)
0101b – 40ns to 50ns (~5m)
0110b – 50ns to 60ns (~6m)
0111b – 60ns to 70ns (~7m)
1000b –1000ns (~100m)
1001b –2000ns (~200m)
1010b – 3000ns (~300m)
1011b ….1111b Reserved, shall not be used
Includes latency of electronics in Active Cable

in.Rsvd_1: Reserved

in.Type_C_to: Type-C plug to Type-C / Captive

00b = Reserved
01b = Reserved
10b = Type-C
11b = Captive

in.Rsvd_2: Reserved

Shall be set to zero

in.VDO_Ver: VDO Version

010b – Version 1.2
000b, 001b, 011 … 111b – Reserved

in.FW_Ver: Firmware Version

0000b..1111b assigned by the VID owner

in.HW_Ver: HW Version

0000b..1111b assigned by the VID owner

5.2.7.8.12 *Vendor Defined Discover ID Response Active Cable VDO 2 Field Names (Rev_3.0)*

in.USB_Gen: USB Gen

0b = Gen 1
1b = Gen 2 or higher
Note: see VDO1 USB Highest Speed for details of Gen supported.

in. USB4_Asymmetric_Mode_Supp:
0 = No
1 = Yes

in.Opt_Isolated_Act_Cable:

0 = No
1 = Yes

in.USB_Lanes_Supp:

0 = One lane
1 = Two lanes

in.USB_3_2_Supp:

0 = USB 3.2 supported
1 = USB 3.2 not supported

in.USB_2_Supp:

0 = USB2.0 supported
1 = USB2.0 not supported

in.USB2_Hub_Hops:

Number of USB2.0 'hub hops' cable consumes.
Shall be set to 0 if USB 2.0 not supported.

in.USB4_Supp:
0 = USB4 supported
1 = USB4 not supported

in.Active_Element:

0 = Active Redriver
1 = Active Retimer

in.Phys_Conn:

0 = Copper
1 = Optical

in.U3ToU0_Trans_Mode:

00: U3 to U0 direct
01: U3 to U0 through U3S

in.U3_CLd_Power:

000: >10mW
001: 5-10mW
010: 1-5mW

011: 0.5-1mW
100: 0.2-0.5mW
101: 50-200µW
110: <50µW
111: ***Reserved***, ***Shall Not*** be used

in.Rsvd_3: Reserved

in.Shutdown_Temp:

The temperature at which the cable will go into thermal shutdown so as not to exceed the allowable plug skin temperature.

in.Max_Oper_Temp:

The maximum internal operating temperature. It may or may not reflect the plug's skin temperature.

5.2.7.8.13 *Vendor Defined Discover ID Response Alternate Mode VDO Field Names*

in.USB_Highest_Speed: USB Highest Speed

000b = [USB 2.0] only
001b = [USB3.2] Gen1 and USB 2.0
010b = [USB3.2] Gen1, Gen2 and USB 2.0
011b = [USB 2.0] billboard only
100b.. 111b = Reserved, shall not be used

in.VBUS_Req: VBUS required

0 = No
1 = Yes

in.VCONN_Req: CONN required

0 = No
1 = Yes

in.VCONN_Pow: VCONN power VCONN power needed by adapter for full functionality

000b = 1W
001b = 1.5W
010b = 2W
011b = 3W
100b = 4W
101b = 5W
110b = 6W
111b = Reserved, shall not be used

in.SS_RX2: SSRX2 Directionality Support (Rev 2.0)

0 = Fixed
1 = Configurable

in.SS_RX1: SSRX1 Directionality Support (Rev 2.0)

0 = Fixed
1 = Configurable

in.SS_TX2: SSTX2 Directionality Support (Rev 2.0)

0 = Fixed
1 = Configurable

in.SS_TX1: SSTX1 Directionality Support (Rev 2.0)

0 = Fixed
1 = Configurable

in.Rsvd: Reserved. Shall be set to zero

in.VDO_Ver: VDO Version (Rev 3.0)

000b – Version 1
001b … 111b – Reserved

in.FW_Ver: Firmware Version

0000b..1111b assigned by the VID owner

in.HW_Ver: HW Version

0000b..1111b assigned by the VID owner

5.2.7.8.14 *Vconn Powered USB Device VDO (Rev 3.0 only)*

in.CT_Supp:

1b – the VPD supports Charge Through
0b – the VPD does not support Charge Through

in.Ground_Impd:

Charge Through Support bit = 1b: Ground impedance through the VPD in 1 mΩ increments. Values less than 10 mΩ are ***Reserved*** and ***Shall Not*** be used.
Charge Through Support bit = 0b: ***Reserved, Shall*** be set to zero

in.VBus_Impd:

Charge Through Support bit = 1b: Vbus impedance through the VPD in 2 mΩ increments. Values less than 10 mΩ are Reserved and ***Shall Not*** be used.
Charge Through Support bit = 0b: ***Reserved, Shall*** be set to zero

in.Rsvd_1: Reserved

in.Charge_Through_Curr_Supp:

Charge Through Support bit = 1b:sop
0b = 3A capable
1b = 5A capable
Charge Through Support bit = 0b: : ***Reserved, Shall*** be set to zero

in.VBus_Max_Vol:

Maximum Cable VBUS Voltage:
00b – 20V
01b – 30V
10b – 40V
11b – 50V

in.Rsvd_2: Reserved

in.VDO_Ver:

Version 1.0 = 000b
Values 001b…111b are ***Reserved*** and ***Shall Not*** be used

in.FW_Ver: Firmware Version

0000b..1111b assigned by the VID owner

in.HW_Ver: HW Version

0000b..1111b assigned by the VID owner

5.2.7.8.15 *Vendor Defined Discover SVIDs Response VDO Field Names*

Do not add prefix to these fields.

SVID_[(obj_num - 1) * 2 + 1]:

SVID n+1 16 bit unsigned integer, assigned by the USB-IF or
0x0000 if this is the last VDO and the Responder supports an odd or even number of SVIDs.

SVID_[(obj_num - 1) * 2]:

SVID n 16 bit unsigned integer, assigned by the USB-IF or
0x0000 if this is the last VDO and the Responder supports an even number of SVIDs.

Where **obj_num** is a number between 1 and **in.DataObjectCount**

5.2.7.8.16 *Vendor Defined Discover Modes Response VDO Field Names*

Do not add prefix to these fields.

To retrieve field value using "Packet And Script Decoded Fields Retrieving Functions", field name format should be:
`Mode X`

To retrieve field value using "Input Context", field name format should be:
`Mode_X`

Where X is a number between 1 and **`in.DataObjectCount`**

5.2.7.8.17 *DisplayPort (VDM)*

The prefix has to be added for all fields related to DisplayPort. All these fields values would get extracted using **`GetHexScriptField()`** function or "**`in`**" context.

Note: The "X" in the prefix (**`OBJX`**) indicates the Data Object Number starting from 2 (e.g. **`in.OBJ2_<field_name>`**)

5.2.7.8.17.1 SOP DisplayPort Capabilities (VDO in Responder Discover Modes VDM) Field Names

`in.OBJX_Port_Cap:` Port Capability.
Length: 2 bits
Possible values:
00: Reserved Value
01: UFP_D-capable
10: DFP_D-capable
11: Both DFP_D and UFP_D-capable

`in.OBJX_Sig_Supp:` Signaling for Transport of DisplayPort Protocol.
Length: 4 bits
Possible values:
xxx1: Supports DP bit rates and electrical settings
xx1x: Reserved
x1xx: Reserved
1xxx: Reserved

`in.OBJX_Recep_Ind:` Receptacle Indication.
Length: 1 bit
Possible values:
0: DisplayPort interface is presented on a USB Type-C Plug
1: DisplayPort interface is presented on a USB Type-C Receptacle

`in.OBJX_USB_r2_0:` USB r2.0 Signaling Not Used.
Length: 1 bit
Possible values:
0: USB r2.0 signaling may be required on A6 – A7 or B6 – B7 while in DisplayPort Configuration
1: USB r2.0 signaling is not required on A6 – A7 or B6 – B7 while in DisplayPort Configuration

in.OBJX_DFP_D_Pins: DFP_D Pin Assignments Supported.
Length: 8 bits
Possible values:
00000000: DFP_D pin assignments are not supported
xxxxxxx1: Reserved
xxxxxx1x: Reserved
xxxxx1xx: Pin assignment C is supported
xxxx1xxx: Pin assignment D is supported
xxx1xxxx: Pin assignment E is supported
xx1xxxxx: Reserved
x1xxxxxx: Reserved
1xxxxxxx: Reserved

in.OBJX_UFP_D_Pins: UFP_D Pin Assignments Supported.
Length: 8 bits
Possible values:
00000000: UFP_D pin assignments are not supported
xxxxxxx1: Reserved
xxxxxx1x: Reserved
xxxxx1xx: Pin assignment C is supported
xxxx1xxx: Pin assignment D is supported
xxx1xxxx: Pin assignment E is supported
xx1xxxxx: Reserved
x1xxxxxx: Reserved
1xxxxxxx: Reserved

in.OBJX_Rsvd: Reserved.
Length: 6 bits

in.OBJX_DPAM_Ver: DPAM version.
Length: 2 bits
Possible values:
00: Version 2.0 or earlier
01: Version 2.1 or higher

5.2.7.8.17.2 SOP' Cable DisplayPort Capabilities (VDO in Responder USB PD Discover Modes VDM) Field Names

in.OBJX_Rsvd_1: Reserved
Length: 2 bits

in.OBJX_Sig_Supp: Signaling for Transport of DisplayPort Protocol.
Length: 4 bits
Possible values:
xxx1: Supports all defined DP bit rates up to HBR3.
xx1x: Supports DP bit rate UHBR10
x1xx: Supports DP bit rate of UHBR20
1xxx: Reserved

in.OBJX_Rsvd_2: Reserved.
Length: 2 bit

in.OBJX_DFP_D_Pins: DP Source Device Pin Assignments Supported.
Length: 8 bits
Possible values:
0C: Pin Assignments C and D are supported
10: USB-C and DP connector Pin Assignment E is supported
All other values are reserved

in.OBJX_UFP_D_Pins: UFP_D Pin Assignments Supported.
Length: 8 bits
Possible values:
0C: Pin Assignments C and D are supported
10: USB-C and DP connector Pin Assignment E is supported
All other values are reserved

in.OBJX_Rsvd_3: Reserved.
Length: 2 bits

in.OBJX_UHBR13_5: UHBR13.5.
Length: 1 bit
Possible values:
0: UHBR13.5 is not supported.
1: UHBR13.5 is supported

in.OBJX_Rsvd_4: Reserved.
Length: 1 bit

in.OBJX_Cable_Type: Active Component.
Length: 2 bits
Possible values:
00: Passive.
01: Active re-timer. 10b = Active re-driver.
11: Optical

in.OBJX_DPAM_Ver: DPAM version.
Length: 2 bits
Possible values:
00: Version 2.0 or earlier
01: Version 2.1 or higher

5.2.7.8.17.3 SOP DisplayPort Status Field Names

in.OBJ2_Conn: DFP_D/UFP_D Connected.
Length: 2 bits
Possible Values:
00: Neither DFP_D nor UFP_D is connected, or Adaptor is disabled
01: DFP_D is connected
10: UFP_D is connected
11: Both DFP_D and UFP_D are connected

in.OBJ2_Pow_Low: Power Low.
Length: 1 bit

Possible values:
0: Adaptor is functioning normally or is disabled
1: Adaptor has detected low power and disabled DisplayPort support

in.OBJ2_Adaptor_Func: Enabled.
Length: 1 bit
Possible values:
0: Adaptor DisplayPort functionality is disabled
1: Adaptor DisplayPort functionality is enabled and operational

in.OBJ2_Multi_Func: Multi-function Preferred.
Length: 1 bit
Possible values:
0: No preference for Multi-function
1: Multi-function is preferred

in.OBJ2_USB_Config: USB Configuration Request.
Length: 1 bit
Possible values:
0: Maintain current configuration
1: Request switch to USB Configuration (if in DisplayPort Configuration)

in.OBJ2_Exit_DP: Exit DisplayPort Mode Request.
Length: 1 bit
Possible values:
0: Maintain current mode
1: Request exit from DisplayPort Mode (if in DisplayPort Mode)

in.OBJ2_HPD_State: HPD State.
Length: 1 bit
Possible values:
0: HPD_Low
1: HPD_High

in.OBJ2_IRQ_HPD: IRQ_HPD.
Length: 1 bit
Possible values:
0: No IRQ_HPD since last status message
1: IRQ_HPD

in.OBJ2_NO_DPAM_SUSPEND: NO_DPAM_SUSPEND.
Length: 1 bit
Possible values:
0: UFP_U/ DP Sink device has no preference for entry into low power state
1: UFP_U/ DP Sink device prefers not to enter low power state

in.OBJ2_Rsvd: Reserved.
Length: 22 bits

5.2.7.8.17.4 SOP' Active Cable DisplayPort Status Update Field Names

in.OBJ2_Rsvd_1: Reserved.
Length: 3 bits

in.OBJ2_Adapter_Func: Enabled.
Length: 1 bit
Possible values:
0: Active cable DP functionality is disabled
1: Active cable DP functionality is enabled and operational

in.OBJ2_Rsvd_2: Reserved.
Length: 28 bits

5.2.7.8.17.5 SOP DisplayPort Configurations Field Names

in.OBJ2_Set_Config: Select Configuration.
Length: 2 bits
Possible values:
00: Set configuration for USB
01: Set configuration for UFP_U as a DP Source device
10: Set configuration for UFP_U as a DP Sink device
11: Reserved

in.OBJ2_Set_Sig: Signaling for transport of DisplayPort Protocol.
Length: 4 bits
Possible values:
0001: Supports all defined DP bit rates up to HBR3 or capability is unknown
0010: Supports DP bit rate UHBR10
0100: Supports DP bit rate of UHBR20
All other values are reserved for higher bit-rates.

in.OBJ2_Rsvd_1: Reserved.
Length: 2 bits

in.OBJ2_Config_UFP_U: Configure UFP_U Pin Assignment.
Length: 8 bits
Possible values:
00000000: De-select pin assignment
00000100: Select pin Assignment C
00001000: Select pin Assignment D
00010000: Select pin Assignment E
All other values are reserved

in.OBJ2_Rsvd_2: Reserved.
Length: 10 bits

in.OBJ2_Cable_UHBR13_5_Support: Cable UHBR13.5 Support.
Length: 1 bit
Possible values:
0 = Not supported or capability is unknown
1 = Supported

in.OBJ2_Rsvd_3: Reserved.
Length: 1 bit

in.OBJ2_ Cable_Type: Cable Active Component.
Length: 2 bits
Possible values:
00: Passive or cable type is unknown
01: Active re-timer.
10: Active re-driver.
11: Optical

in.OBJ2_DPAM_Ver: DPAM Version.
Length: 2 bits
Possible values:
00: Version 2.0 or earlier
01: Version 2.1 or higher

5.2.7.8.17.6 SOP' Active Cable DisplayPort Configurations Field Names

in.OBJ2_Set_Config: Select Configuration.
Length: 2 bits
Possible values:
00: Select configuration for USB
01: Select configuration for active cable as DP Source device
10: Select configuration for active cable as a DP Sink device
11: Reserved

in.OBJ2_Set_Sig: Signaling for transport of DisplayPort Protocol.
Length: 4 bits
Possible values:
0000: Bit rate is unspecified
0001: Select DP bit rates and electrical settings
All other values are reserved

in.OBJ2_Rsvd_1: Reserved.
Length: 2 bits

in.OBJ2_Config_UFP_U: Configure UFP_U Pin Assignment.
Length: 8 bits
Possible values:
00000000: De-select pin assignment
00000100: Select pin Assignment C
00001000: Select pin Assignment D
00010000: Select pin Assignment E
All other values are reserved

in.OBJ2_Rsvd_2: Reserved.
Length: 16 bits

5.2.7.9 Battery Status Message

Note: Battery Status packet prefix is always "obj1_"

in.Rsvd: Reserved

in.Battery_Info: Bit field information about battery as per spec.

in.Battery_PC: Battery's state of charge.

5.2.7.10 Alert Message

Note: Alert packet prefix is always "obj1_"

in.Event_Type: Extended Alert Event Type
Length: 4 bits
Possible values:
0000: Reserved
0001: Power state change (DFP only)
0010: Power button press (UFP only)
0011: Power button release (UFP only)
0100: Controller initiated wake e.g., Wake on Lan (UFP only)
All other values are reserved

in.Rsvd: Reserved
Length: 12 bits

in.Hot_Swap_Batteries: 4-bit bit field, When Battery Status Change bit is set, indicates which Hot Swappable Batteries have had a status change. lsb corresponds to Battery 0 and msb corresponds to Battery 3

in.Fixed_Batteries: 4-bit bit field, When Battery Status Change bit is set, indicates which Fixed Batteries have had a status change. lsb corresponds to Battery 0 and msb corresponds to Battery 3

in.Alert_Type: 8-bit bit field as follows

Bit representations:
0: Reserved and shall be set to zero
1: Battery Status Change Event (attach/detach/charging/discharging/idle)
2: OCP event when set (Source only, for Sink Reserved and shall be set to zero)
3: OTP event when set
4: Operating Condition Change when set
5: Source Input Change Event when set
6: OVP event when set (Sink only, for Source Reserved and shall be set to zero)
7: Extended Alert Event

5.2.7.11 Get Country Info Message

Note: Get Country Info packet prefix is always "obj1_"

in.Rsvd: Reserved

in.Country_Code:

Alpha-2 Country
Code defined by [ISO 3166]

5.2.7.12 Enter_USB Message

Note: Enter_USB messages do not require an OBJx_ prefix, as they only contain a single Data Object.

in.Rsvd_1: Shall be set to zero.

in.Host_Present: Connected to a host

in.TBT_Support: TBT3 is supported by the host's USB4 Connection Manager

in.DP_Support: USB4 DP tunneling supported by the host

in.PCIe_Support: USB4 PCIe tunneling supported by the host

in.Cable_Current:

00b = V_{BUS} is not supported
10b = 3A
11b = 5A

in.Cable_Type:

00b: Passive
01b: Active Re-timer
10b: Active Re-driver
11b: Optically Isolated

in.Cable_Speed:

000b: USB 2.0 only, no SuperSpeed support
001b: USB 3.2 Gen1
010b: USB 3.2 Gen2 and [USB4] Gen2
011b: USB4 Gen3
100b: USB4 Gen4

in.Rsvd_2: Shall be set to zero.

in.USB3_DRD: Capable of operating as a USB 3.2 Device

in.USB4_DRD: Capable of operating as a USB4 Device

in.Rsvd_3: Shall be set to zero.

in.USB_Mode:

000b: USB 2.0

001b: USB 3.2
010b: USB4

in.Rsvd_4: Shall be set to zero.

5.2.7.13 EPR Request

Please refer to section 5.2.7.4 (Request Packet Data Field Names) for RDO, and section 5.2.7.3.1 (Source Capabilities Message) for PDO.

5.2.7.14 EPR Mode

in.Reserved: Reserved field

in.Data: Provides additional information.

Data = EPR Sink Operational PDP, If Action is _PD_EPR_MODE_ACTION_ENTER

Data = 0, if Action is
_PD_EPR_MODE_ACTION_ENTER_ACK
_PD_EPR_MODE_ACTION_ENTER_SUCCEED
_PD_EPR_MODE_ACTION_EXIT

If Action = _PD_EPR_MODE_ACTION_ENTER_FAILED
Data = 0: Unknown cause
Data = 1: Cable not EPR capable
Data = 2: Source failed to become V_{CONN} source
Data = 3: EPR Mode Capable bit not set in RDO
Data = 4: Source unable to enter EPR Mode at this time
Data = 5: EPR Mode Capable bit not set in PDO

in.Action: Describes the action that is to be taken by the recipient of the EPR_Mode message:

_PD_EPR_MODE_ACTION_ENTER = 1
_PD_EPR_MODE_ACTION_ENTER_ACK = 2
_PD_EPR_MODE_ACTION_ENTER_SUCCEED= 3
_PD_EPR_MODE_ACTION_ENTER_FAILED = 4
_PD_EPR_MODE_ACTION_EXIT = 5

5.2.7.15 Source Info

in.Port_Reported_PDP: Power the port is actually advertising.

in.Port_Present_PDP: Power the port is presently capable of supplying.

in.Port_Maximum_PDP: Power the port is designed to supply.

in.Reserved: Reserved.

in.Port_Type: 0 = Shared capacity port.
1 = Assured capacity port

5.2.7.16 Revision

in.Reserved: Reserved data.

in.Version_Minor: Version minor.

in.Version_Major: Version major.

in.Revision_Minor: Revision minor.

in.Revision_Major: Revision major.

5.2.7.17 Extended Control

in.Type: Extended control message type. The values can be 1 to 4. The rest values are reserved.

in.Data: Corresponding data.

5.2.7.18 Source Capabilities Extended Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.VID: Vendor ID (assigned by the USB-IF)

in.PID: Product ID (assigned by the manufacturer)

in.XID: Value provided by the USB-IF to assign to product

in.FW_Ver: Firmware version number

in.HW_Ver: Hardware version number

in.Vol_Reg: Voltage Regulation

in.Holdup_Time: Holdup Time

in.Compliance: Compliance, the standards the Source is compliant with

in.Touch_Curr: Touch Current

in.Peak_Curr_1: Peak Current 1

in.Peak_Curr_2: Peak Current 2

in.Peak_Curr_3: Peak Current 3

in.Touch_Temp: Temperature conforms to:

0 = [IEC 60950-1] (default)
1 = [IEC 62368-1] TS1
2 = [IEC 62368-1] TS2
Note: All other values Reserved

in.Source_Inputs: Source Inputs identifies the possible inputs that provide power to the Source

in.Hot_Swap_Batteries: The number of hot swappable batteries that source supports

in.Fixed_Batteries: The number of fixed batteries that source supports

5.2.7.19 Status Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Internal_Temp: Source or Sink's internal temperature in degrees centigrade

Special values:
0 = feature not supported
1 = temperature is less than 2°C

in.Flags: (non-SOP only)

bit0: Thermal Shutdown
bit1..7: Reserved

in.Src_Input: Indicates which supplies are presently powering the Source

in.Fixed_Battery_Input: Fixed Battery Input presents

in.Hot_Swap_Battery_Input: Hot Swappable Battery Input presents

in.Event_Flags:

bit 0: Reserved
bit 1: OCP
bit 2: OTP
bit 3: OVP
bit 4: CF mode when set, CV mode when cleared
bit 5..7:Reserved

in.Temp_Status:

bit 0: Reserved
bit 1..2:
00 – Not Supported
01 – Normal
10 – Warning
11 – Over temperature
bit 3..7: Reserved

in.Power_Status:

bit 0: Reserved
bit 1: Source power limited due to cable supported current
bit 2: Source power limited due to insufficient power available while sourcing other ports
bit 3: Source power limited due to insufficient external power
bit 4: Source power limited due to Event Flags in place (Event Flags must also be set)
bit 5: Source power limited due to temperature
bit 6..7: Reserved

in.Power_State_Change:

bit 0..2: New power state
bit 3..5: New power state indicator
bit 6..7: Reserved

5.2.7.20 Get Battery Cap Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Hot_Swap_Ref: Requested hot swappable batteries

in.Fixed_Ref: Requested fixed batteries

5.2.7.21 Get Battery Status Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Hot_Swap_Ref: Requested hot swappable batteries

in.Fixed_Ref: Requested fixed batteries

5.2.7.22 Battery Capabilities Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.VID: Vendor ID (assigned by the USB-IF)

in.PID: Product ID (assigned by the manufacturer)

in.Design_Cap: Battery's Design Capacity

in.Last_Full_Charge_Cap: Battery's Last Full Charge Capacity

in.Battery_Type: Bit field indicating the type of battery

5.2.7.23 Get Manufacturer Info Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Target: Manufacturer Info Target

in.Hot_Swap_Ref: If **Target** is battery (01b) then this shall contain the hot swappable battery number reference, which is the number of the Battery indexed from zero.

in.Fixed_Ref: If **Target** is battery (01b) then this shall contain the fixed battery number reference, which is the number of the Battery indexed from zero.

5.2.7.24 Manufacturer Info Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.VID: Vendor ID (assigned by the USB-IF)

in.PID: Product ID (assigned by the manufacturer)

in.String: Vendor defined byte array

5.2.7.25 Security Request Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.DataLen: Length of the data in this message

in.Security_Req_Data: Security request data

5.2.7.26 Security Response Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.DataLen: Length of the data in this message

in.Security_Resp_Data: Security response data

5.2.7.27 Firmware Update Request Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.DataLen: Length of the data in this message

in.FW_Update_Req_Data: Firmware update request data

5.2.7.28 Firmware Update Response Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.DataLen: Length of the data in this message

in.FW_Update_Resp_Data: Firmware update response data

5.2.7.29 PPS Status Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Output_Voltage:

Source's output voltage in 20mV units.
When set to 0xFFFF, the Source does not support this field.

in.Output_Current:

Source's output current in 50mA units.
When set to 0xFF, the Source does not support this field.

in.Rsvd_1: Reserved
in.PTF:

PTF: 00 – Not Supported
PTF: 01 – Normal
PTF: 10 – Warning
PTF: 11 – Over temperature

in.OMF:

OMF set when operating in Current Limit mode, and cleared when operating in Constant Voltage mode.

in.Rsvd_2: Reserved

5.2.7.30 Country Info Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Country_Code:

Alpha-2 Country Code received in the corresponding Get_Country_Info Message

in.Rsvd: Reserved

in.Specific_Data:

0…256 bytes of content defined by the country's authority.

5.2.7.31 Country Code Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.Num:

Number of country codes in the message

in.Country_Code:

Byte stream of Alpha-2 Country Code

5.2.7.32 Sink Capabilities Extended Message

Note: Extended Messages do not have OBJx_ prefix, as there is no definition of Data Object. Neither in Chunked nor in Unchunked mode.

in.VID: Vendor ID (assigned by the USB-IF)

in.PID: Product ID (assigned by the manufacturer)

in.XID: Value provided by the USB-IF assigned to the product

in.FW_Ver: Firmware version number

in.HW_Ver: Hardware version number

in.SKEDB_Version:

SKEDB Version (not the specification Version):
Version 1.0 = 1
Values 0 and 2-255 are Reserved and Shall Not be used

in.Load_Step:

bit 0..1:
00b: 150mA/µs Load Step (default)
01b: 500mA/µs Load Step
11b…10b: Reserved and Shall Not be used

bit 2..7: Reserved

in.Load_Characteristics:

bit 0..4: Percent overload in 10% increments. Values higher than 25 (11001b) are clipped to 250%. 00000b is the default.

bit 5..10: Overload period in 20ms when bits 0-4 non-zero.
bit 11..14: Duty cycle in 5% increments when bits 0-4 are non-zero
bit 15: Can tolerate VBUS Voltage droop

in.Compliance:
bit 0: Requires LPS Source when set
bit 1: Requires PS1 Source when set
bit 2: Requires PS2 Source when set
bit 3..7: Reserved

in.Touch_Temp:
Temperature conforms to:
0 = Not Applicable
1 = [IEC 60950-1] (default)
2 = [IEC 62368-1] TS1
3 = [IEC 62368-1] TS2
Note: All other values Reserved

in.Hot_Swapp_Batt_Num: The number of hot swappable batteries that source supports

in.Fixed_Batt_Num: The number of fixed batteries that source supports

in.Sink_Modes:

bit 0: 1: PPS charging supported
bit 1: 1: VBUS powered

bit 2: 1: Mains powered
bit 3: 1: Battery powered
bit 4: 1: Battery essentially unlimited
bit 5..7: Reserved

in.Sink_Min_PDP:

bit 0..6:
The Minimum PDP required by the Sink to operate without consuming any power from its Battery(s) should it have one

bit 7: Reserved

in.Sink_Op_PDP:

bit 0..6: The PDP the Sink requires to operate normally
bit 7: Reserved

in.Sink_Max_PDP:

bit 0..6: The Maximum PDP the Sink can consume to operate and charge its Battery(s) should it have one.
bit 7: Reserved

in.EPR_Sink_Min_PDP:

The Minimum PDP required by the EPR Sink to operate without consuming any power from its Battery(s) should it have one.

in.EPR_Sink_Op_PDP:

The PDP the EPR Sink requires to operate normally. For Sinks with a Battery, it is the PDP Rating of the charger supplied with it or recommended for it.

in.EPR_Sink_Max_PDP:

The Maximum PDP the EPR Sink can consume to operate and charge its Battery(s) should it have one.

5.2.7.33 Vendor Defined Extented

It has an unstructured VDM Header. Please refer to section .5.2.7.8.1

5.2.7.34 Power Delivery Events

In the case of an event packet, in addition to **in.PacketType**, the following fields are available:

in.EventType: Indicates the type of event, possible values:

_PD_EVENT_TYPE_FAST_POWER_ROLE_SWAP => Fast Role Swap Signal Event

5.2.7.35 Unknown Packet

If message could not be decoded, it is an unknown packet containing following fields:

in.HasPacketFraming

if **in.HasPacketFraming == 1, in.ProbableMessageType** contains probable message type

5.2.8 PD Transaction-specific Set of Members

PD Transactions are exactly similar to underlying packets.
Please note that when a transaction contains more than one chunked packet underneath, it represents the complete **unchunked** packet.

5.2.9 PD Transfer-specific Set of Members

5.2.9.1 USB Authentication (available in PD Security Messages)

Since both request and response of a PD Security set of messages are available in one context, to prevent conflict the respective header fields are being separated through a prefix. Request header fields are prefixed with **Req_** and Response ones are prefixed with **Resp_**. The payload data has its own prefix per Message type as follows.

5.2.9.1.1 *Request Header fields*

in.Req_Version: USB Type-C Authentication Specification Version

in.Req_Msg_Type: Authentication Message Type

Available constants:
_AUTH_MSG_REQ_GET_DIGESTS
_AUTH_MSG_REQ_GET_CERTIFICATE
_AUTH_MSG_REQ_CHALLENGE

in.Req_Param1: Message type specific parameter 1 as per USB Type-C Authentication specifications

in.Req_Param2: Message type specific parameter 2 as per USB Type-C Authentication specifications

5.2.9.1.2 *GET_CERTIFICATE Request Payload fields*

in.GetCertificate_Offset: Offset in bytes from the start of the Certificate Chain to where the read request begins

in.GetCertificate_Length: Length in bytes of the read request

5.2.9.1.3 *CHALLENGE Request Payload fields*

in.Challenge_Nonce: Random 32-byte nonce chosen by the Authentication Initiator

5.2.9.1.4 *Response Header fields*

in.Resp_Version: USB Type-C Authentication Specification Version

in.Resp_Msg_Type: Authentication Message Type

Available constants:
_AUTH_MSG_RESP_DIGESTS
_AUTH_MSG_RESP_CERTIFICATE

_AUTH_MSG_RESP_CHALLENGE_AUTH
_AUTH_MSG_RESP_ERROR

in.Resp_Param1: Message type specific parameter 1 as per USB Type-C Authentication specifications

in.Resp_Param2: Message type specific parameter 2 as per USB Type-C Authentication specifications

5.2.9.1.5 *DIGESTS Response Payload fields*

in.Digests_Count: Count of digests that this message contains

in.Digests_Digest_0: First 32-byte SHA-256 digest of the first Certificate Chain

in.Digests_Digest_1: Second 32-byte SHA-256 digest of the first Certificate Chain
.
.
.
in.Digests_Digest_[Digests_Count-1]: Last 32-byte SHA-256 digest of the first Certificate Chain

5.2.9.1.6 *CERTIFICATE Response Payload fields*

in.Certificate_CertChain: Requested contents of target Certificate Chain

in.Certificate_CertChainLength: Length of the Certificate Chain

5.2.9.1.7 *CHALLENGE_AUTH Response Payload fields*

in.Challenge_Auth_MinProtocolVersion: Minimum protocol version supported by this Device

in.Challenge_Auth_MaxProtocolVersion: Maximum protocol version supported by this Device

in.Challenge_Auth_Capabilities: As per spec

in.Challenge_Auth_Reserved: Reserved

in.Challenge_Auth_CertChainHash: 32-byte SHA256 hash of the Certificate Chain used for Authentication

in.Challenge_Auth_Salt: 32-byte value chosen by the Authentication Responder

in.Challenge_Auth_ContextHash: As per spec

in.Challenge_Auth_Signature: As per spec

5.2.10 CC Packet-specific Set of Members

Note 1: All field values can be retrieved via "Input Context" or "Packet And Script Decoded Fields Retrieving Functions".

5.2.10.1 Information Available Only by Context

Note 1: If these variables are used in Exerciser mode, Left means DUT and Right means Exerciser

in.CC1_Pins_Changed: Indicates if CC1 pin state changed or not

0: State not changed
1: State changed

in.Ad_Curr_Changed: Indicates whether Advertised Current Changed or not

0: State not changed
1: State changed

in.Left_CurrState: Indicates Left port's Current State
in.Left_PrevState: Indicates Left port's Previous State
in.Left_MiddleState: In case the port transitioned into a middle state in its transition from previous to current state this field will provide that. Otherwise, its value is _INVALID_STATE (0).

in.Right_CurrState: Indicates Right port's Current State
in.Right_PrevState: Indicates Right port's Previous State
in.Right_MiddleState: In case the port transitioned into a middle state in its transition from previous to current state this field will provide that. Otherwise, its value is _INVALID_STATE (0).

Possible state values are:

_SRC_UNATTACHED => Unattached.SRC
_SRC_ATTACH_WAIT => AttachWait.SRC
_SRC_ATTACHED => Attached.SRC
_SRC_TRY => Try.SRC
_SRC_TRY_WAIT => TryWait.SRC

_SNK_UNATTACHED => Unattached.SNK
_SNK_ATTACH_WAIT => AttachWait.SNK
_SNK_ATTACHED => Attached.SNK
_SNK_TRY => Try.SNK
_SNK_TRY_WAIT => TryWait.SNK

_AUDIO_ACCESSORY_ATTACHED => AudioAccessory
_DEBUG_ACCESSORY_ATTACHED => DebugAccessory.SRC
_POWERED_ACCESSORY_ATTACHED => PoweredAccessory

_AUDIO_ADAPTER => auxiliary state to show the audio device
_DEBUG_ADAPTER => DebugAccessory.SNK

_DISABLED => the port is disabled
_UNDEFINED_STATE => the state is unknown
_XXX_ATTACHED => either Attached.Source or Attached.Sink, but unclear which one

in.Left_StateChange: Indicates if left port (DUT port in exerciser mode) state changed or not

0: State not changed
1: State changed

in.Left_CC1StateChange: Indicates if CC1 port state changed or not

0: State not changed
1: State changed

in.Left_CC2StateChange: Indicates if CC2 port state changed or not

0: State not changed
1: State changed

in.Right_StateChange: Indicates if right port (Exerciser port in exerciser mode) state changed or not

0: State not changed
1: State changed

in.Right_CC1StateChange: Indicates if CC1 port state changed or not

0: State not changed
1: State changed

in.Right_CC2StateChange: Indicates if CC2 port state changed or not

0: State not changed
1: State changed

5.2.10.2 CC Message

in.CC1_Pins:Indicates CC1 pin connection status

0: Not Connected
1: Connected

in.Ad_Curr: Current Rp value that can be any of the following:

_CC_RP_NONE
_CC_RP_DEF
_CC_RP_15A
_CC_RP_30A

in.Left_CC1: Indicates current left CC1 port state

in.Left_CC2: Indicates current left CC2 port state

in.Right_CC1: Indicates current right CC1 port state

in.Right_CC2: Indicates current right CC2 port state

Port states can contain following values:

```
_CC_PORT_STATE_UNKNOWN        => state is unknown
_CC_PORT_STATE_OPEN           => port is disabled and state is open
_CC_PORT_STATE_RP             => Rp
_CC_PORT_STATE_RD             => Rd
_CC_PORT_STATE_RA             => Ra
_CC_PORT_STATE_VCONN          => VConn
_CC_PORT_STATE_RD_RA          => state is unclear; it is either Rd or Ra
_CC_PORT_STATE_RD_RA_OPEN     => state is unclear; it is Rd, Ra or Open
_CC_PORT_STATE_RP_VCONN       => state is unclear; it is either Rp or VConn
_CC_PORT_STATE_RP_VCONN_OPEN => state is unclear; it is Rp, VConn or Open
```

5.2.11 USB4 Operation-specific Set of Members

in.Type: Type of operation. May be one of the values in the table below.

Operation type	Description
_USB4_OP_TYPE_DROM_READ	DROM Read Router Operation
_USB4_OP_TYPE_QUERY_DP_RESOURCE_AVAILABILITY	Query DP Resource Availability Router Operation
_USB4_OP_TYPE_ALLOCATE_DP_RESOURCE	Allocate DP Resource Router Operation
_USB4_OP_TYPE_DEALLOCATE_DP_RESOURCE	De-allocate DP Resource Router Operation
_USB4_OP_TYPE_NVM_WRITE	NVM Write Router Operation
_USB4_OP_TYPE_NVM_AUTHENTICATE_WRITE	NVM Authenticate Write Router Operation
_USB4_OP_TYPE_NVM_READ	NVM Read Router Operation
_USB4_OP_TYPE_NVM_SET_OFFSET	NVM Set Offset Router Operation
_USB4_OP_TYPE_GET_NVM_SECTOR_SIZE	Get NVM Sector Size Router Operation
_USB4_OP_TYPE_GET_PCIE_DOWNSTREAM_ENTRY_MAPPING	Get PCIe Downstream Entry Mapping Router Operation
_USB4_OP_TYPE_GET_CAPABILITIES	Get Capabilities Router Operation
_USB4_OP_TYPE_SET_CAPABILITIES	Set Capabilities Router Operation
_USB4_OP_TYPE_BUFFER_ALLOCATION_REQUEST	Buffer Allocation Request Router Operation
_USB4_OP_TYPE_BLOCK_SIDEBAND_PORT_OPERATIONS	Block Sideband Port Operations Router Operation
_USB4_OP_TYPE_UNBLOCK_SIDEBAND_PORT_OPERATIONS	Unblock Sideband Port Operations Router Operation

_USB4_OP_TYPE_GET_CONTAINER_ID	Get Container-ID Router Operation
_USB4_OP_TYPE_UNKNOWN	Unknown Operation

in.IsBonded: Gets a value indicating whether an operation is transferred on a bonded link or not. Value **in.IsBonded** = 0 means that Lanes are unbonded, **in.IsBonded** = 1 means that Lanes are bonded.

in.LaneNumber: Gets the number of lane on which operation is transferred (or started, in Bonded mode). Value **in.LaneNumber** = 0 means Lane 0, **in.LaneNumber** = 1 means Lane 1.

in.TopologyId: Topology ID value for this USB4 operation.

in.AdapterNum: Adapter Number value for this USB4 operation.

in.StatusIsSet: Indicates Status can be retrieved from this USB4 operation.

in.Status: Status value for this USB4 operation.

in.AddressIsSet: Indicates Address can be retrieved from this USB4 operation.

in.Address: Address value for this this USB4 operation.

in.DataSizeIsSet: Indicates DataSize can be retrieved from this USB4 operation.

in.DataSize: Data Size value for this this USB4 operation.

in.EntryNum: Size of EntryTypes array.
Note: Valid for DROM Read router operations only, undefined for other operation types.

in.EntryTypes: Array that contains Entry Type values for this USB4 DROM Read router operation.
Note: Valid for DROM Read router operations only, undefined for other operation types.
Each element may have one of the values in the table below.

Entry type	Description
_USB4_OP_DROM_READ_ENTRY_TYPE_USB4_HEADER_WITH_PADDING	USB4 DROM Header with padding
_USB4_OP_DROM_READ_ENTRY_TYPE_TBT3_ID_AND_HEADER	TBT3 DROM ID and Header
_USB4_OP_DROM_READ_ENTRY_TYPE_UNUSED_ADAPTER_ENTRY	Unused Adapter Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_DP_ADAPTER_ENTRY	DP Adapter Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_TBT3_LANE_ADAPTER_ENTRY	Lane Adapter Entry (TBT3 only)
_USB4_OP_DROM_READ_ENTRY_TYPE_TBT3_PCIE_US_ADAPTER_PORT_ENTRY	PCIe Upstream Adapter Entry (TBT3 only)
_USB4_OP_DROM_READ_ENTRY_TYPE_TBT3_PCIE_DS_ADAPTER_PORT_ENTRY	PCIe Downstream Adapter Entry (TBT3 only)
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_RESERVED_ENTRY	Reserved Generic Entry

_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_VENDOR_SPECIFIC_ENTRY	Vendor Specific Generic Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_ASCII_VENDOR_NAME_ENTRY	ASCII Vendor Name Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_ASCII_MODEL_NAME_ENTRY	ASCII Model Name Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_TMU_MINIMUM_REQUESTED_MODE_ENTRY	TMU Minimum Requested Mode Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_PRODUCT_DESCRIPTOR_ENTRY	Product Descriptor Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_SERIAL_NUMBER_ENTRY	Serial Number Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_USB_PORT_MAPPING_ENTRY	USB Port Mapping Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_UTF16_VENDOR_NAME_ENTRY	UTF16 Vendor Name Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_GENERIC_UTF16_MODEL_NAME_ENTRY	UTF16 Model Name Entry
_USB4_OP_DROM_READ_ENTRY_TYPE_UNKNOWN	Entry Type Unknown

`in.PayloadLength`: Length (in bytes) of the operation payload. If zero, operation does not have any payload. Care should be taken not to request payload for the operations that moved large amounts of data.

`in.Payload`: Bit source of the operation payload. **Note**: You can extract any necessary information using the **GetNBits(), NextNBits()**, or **PeekNBits()** function. (Refer to section 14 in the USBScriptDecodeManual.pdf.)

Note: Refer to the section Script decoded fields retrieving functions for information about getting the script decoded values at the USB4 Operation level.

6 Verification Script Engine Output Context Members

All verification scripts have output contexts, special structures that can be used inside of the application. The scripts fill their members. The verification script output contexts have only one member:

`out.Result`: Result of the whole verification program defined in the verification script.

This member can have six values:

- **_VERIFICATION_PROGRESS**: Set by default when a script starts running.
- **_VERIFICATION_PASSED**
- **_VERIFICATION_PASSED_WARNING**
- **_VERIFICATION_PASSED_WITH_NOTE**
- **_VERIFICATION_NOT_APPLICABLE**
- **_VERIFICATION_FAILED**

The last five values should be set if you decide that a recorded trace does (or does not) satisfy the imposed verification conditions. In these cases, the verification script stops running.

If you do not specify any of those values, the result of script execution is set to VERIFICATION_FAILED at exit.

Note: If you do not care about the result of script running, call the function ScriptForDisplayOnly once before stopping the script. The result is **DONE**.

7 Verification Script Engine Events

VSE defines a large group of trace "events" that can be passed to a verification script for evaluating, retrieving, and displaying some contained information. The information about the type of event can be seen in the **in.TraceEvent** input context member.

The application defines a set of VSE constants for Trace Event types. Those constants should be used by the scripts for determining the event types.

See the topic "Sending Functions" for details about how to specify the events to send to verification scripts.

Packet Level Events

The table below describes the current list of packet level (lowest level of transactions) events and values (application-defined constants) for **in.TraceEvent**:

Types of packets	**in.TraceEvent**
USB2 packet events	
All USB2 packet events	_USB2_PACKET
USB2 Bus conditions	_USB2_BUS_CONDITION
USB2 Start Of Frame packets	_USB2_SOF
USB2 Token packets	_USB2_TOKEN
USB2 Data packets	_USB2_DATA
USB2 Handshake packet	_USB2_HSK
USB2 SPLIT packets	_USB2_SPLIT
USB Full Speed Hub prefix packets	_USB2_PRE
USB2 Link Power Management extended token	_USB2_LPM
All other USB2 packets	_USB2_OTHER
USB3 packet events	
All USB3 packet events	_USB3_PACKET
TSEQ Training Sequence Ordered Sets	_USB3_TSEQ
TS1 Training Sequence Ordered Sets	_USB3_TS1
TS2 Training Sequence Ordered Sets	_USB3_TS2
TS1A Training Sequence Ordered Sets	_USB3_TS1A
TS1B Training Sequence Ordered Sets	_USB3_TS1B
Low Frequency Periodic Signaling	_USB3_LFPS
Interpacket Symbols	_USB3_IPS
Link Commands	_USB3_LINK_CMD
SKIP Sequences	_USB3_SKP_SEQ
Logical Idle Symbols	_USB3_SYMBOL_IDLE
Isochronous Timestamp packets	_USB3_ITP_PKT
Link Management packets	_USB3_LMP_PKT
Transaction packets	_USB3_TP_PKT
Data Packets	_USB3_DP_PKT
Data Packet headers	_USB3_DPH_PKT (only for traces before software version 3.71)
Data payload packets	_USB3_DPP_PKT (only for traces before software version 3.71)
Termination State	_USB3_TERM_STATE
LTSSM State	_USB3_LTSSM_STATE

Compliance Packet	_USB3_CP
VBUS state Packet	_USB2_VBUS_STATE
SSP Sync Packet	_USB3_SYNC
SSP SDS Packet	_USB3_SDS
All other USB3 packets	_USB3_OTHER_PKT
USB4 SB packet events	
All USB4 SB packet events	_USB4_SB_PACKET
AT Transactions	_USB4_SB_AT
RT Transactions	_USB4_SB_RT
LT Transactions	_ USB4_SB_LT
ELT Transactions	_USB4_SB_ELT
SB IPS packets	_USB4_SB_IPS
Aborted packets	_USB4_SB_ABORTED
Connect Event packets	_USB4_SB_CONNECT_EVENT
Disconnect Event packets	_USB4_SB_DISCONNECT_EVENT
USB4 HS packet events	
All USB4 HS packet events	_USB4_HS_PACKET
All USB4 HS TLP packet events	_USB4_HS_TLP_PACKET
All USB4 HS Control packet events	_USB4_HS_CONTROL_PACKET
All USB4 HS Link Management packet events	_USB4_HS_LINK_MANAGEMENT_PACKET
All USB4 HS Time Sync packet events	_ USB4_HS_TIME_SYNC_PACKET
All USB4 HS Ordered Set packet events	_ USB4_HS_OS_PACKET
Symbol Lock Ordered Sets 1	_USB4_HS_SLOS1
Symbol Lock Ordered Sets 2	_USB4_HS_SLOS2
TS1 Training Sequence Ordered Sets	_USB4_HS_TS1
TS2 Training Sequence Ordered Sets	_USB4_HS_TS2
CL1_REQ Ordered Sets	_USB4_HS_CL1_REQ
CL2_REQ Ordered Sets	_USB4_HS_CL2_REQ
CL1_ACK Ordered Sets	_USB4_HS_CL1_ACK
CL2_ACK Ordered Sets	_USB4_HS_CL2_ACK
CL0s_ACK Ordered Sets	_USB4_HS_CL0s_ACK
CL_NACK Ordered Sets	_USB4_HS_CL_NACK
CL_OFF Ordered Sets	_USB4_HS_CL_OFF
CL_WAKE1 Ordered Sets	_USB4_HS_CL_WAKE1
CL_WAKE2 Ordered Sets	_USB4_HS_CL_WAKE2

CL_WAKE1-2 Ordered Sets	_USB4_HS_ALT_CL_WAKE
TSN Training Sequence Ordered Sets	_USB4_HS_TSN
DESKEW Ordered Sets	_USB4_HS_DESKEW
SKIP Ordered Sets	_USB4_HS_SKIP
Follow Up Packets	_USB4_HS_FOLLOW_UP
Inter-Domain Time Stamp Packets	_USB4_HS_INTER_DOMAIN_TIME_STAMP
Idle Packets	_USB4_HS_IDLE
Credit Grant Packets	_USB4_HS_CREDIT_GRANT
Path Credit Sync Packets	_USB4_HS_PATH_CREDIT_SYNC
Shared Buffers Credit Sync Packets	_USB4_HS_SHARED_BUFFERS_CREDIT_SYNC
Read Requests	_USB4_HS_READ_REQUEST
Read Responses	_USB4_HS_READ_RESPONSE
Write Requests	_USB4_HS_WRITE_REQUEST
Write Responses	_USB4_HS_WRITE_RESPONSE
Notification Packets	_USB4_HS_NOTIFICATION
Notification Acknowledgment Packets	_USB4_HS_NOTIFICATION_ACKNOWLEDGEMENT
Hot Plug Event Packets	_USB4_HS_HOT_PLUG_EVENT
Inter-Domain Requests	_USB4_HS_INTER_DOMAIN_REQUEST
Inter-Domain Responses	_USB4_HS_INTER_DOMAIN_RESPONSE
Tunneled Packets	_USB4_HS_TUNNELED_PKT
Interpacket Symbols	_USB4_HS_IPS
Unidentified Symbols Sequences	_USB4_HS_USS
Low Frequency Periodic Signaling	_USB4_HS_LFPS
E2E Credit Grant Packets	_USB4_HS_E2E_CREDIT_GRANT
E2E Credit Sync Packets	_USB4_HS_E2E_CREDIT_SYNC
Filtered Packets	_USB4_HS_FILTERED
START_RS_FEC Packets	_USB4_HS_START_RS_FEC
LLSM State	_USB4_LLSM_STATE
Gen 4 TS1	_USB4_G4_TS1
Gen 4 TS2	_USB4_G4_TS2
Gen 4 TS2.clksw	_USB4_G4_TS2CLKSW
Gen 4 TS3	_USB4_G4_TS3
Gen 4 TS4	_USB4_G4_TS4
Gen 4 DESKEW	_USB4_G4_DESKEW
Gen 4 TSNOS	_USB4_G4_TSNOS
Gen 4 IGNORE	_USB4_G4_IGNORE
Gen 4 UNBOND	_USB4_G4_UNBOND

Gen 4 CL0s_EXIT	_USB4_G4_CL0S_EXIT
Gen 4 CL_OFF	_USB4_G4_CL_OFF
Power Delivery Packets	_PD_PACKET
Power Delivery Events	_PD_EVENT
Power Delivery Transactions	_PD_TRA
Power Delivery Transfers	_PD_XFER

7.1 Transaction Level Events

The table below describes the current list of transaction level events and values (application-defined constants) for **in.TraceEvent**:

Types of transactions	**in.TraceEvent**
Setup transactions	_T_SETUP
In transactions	_T_IN
Out transactions	_T_OUT
All other transaction types	_T_OTHER

7.2 Split Transaction Level Events

The table below describes the current list of split transaction level events and values (application-defined constants) for **in.TraceEvent**:

Types of transactions	**in.TraceEvent**
All split transactions	_SPLIT_TRA

7.3 Transfer Level Events

The table below describes the current list of transfer level events and values (application-defined constants) for **in.TraceEvent**:

Types of transfers	**in.TraceEvent**
Control transfers	_X_CONTROL
Isochronous transfers	_X_ISO
Bulk transfers	_X_BULK
Interrupt transfers	_X_INTERRUPT
All other (undefined) transfer types	_X_OTHER

7.4 SCSI Operation Level Events

The table below describes the current list of SCSI operation level events and values (application-defined constants) for **in.TraceEvent**:

Types of transactions	**in.TraceEvent**
All SCSI operations	_SCSI_OPR

7.5 PHY Transaction Level Events

The table below describes the current list of PHY Transaction level events and values (application-defined constants) for **in.TraceEvent**:

Types of transactions	**in.TraceEvent**
SCD1 Transaction	_P_SCD1
SCD2 Transaction	_P_SCD2
PHY Capability 5G Transaction	_P_PHY_CAP_5
PHY Capability 10G Transaction	_P_PHY_CAP_10
PHY Ready Transaction	_P_PHY_READY

8 Sending Functions

This topic contains information about the special group of VSE functions that specify the events the verification script should expect to receive.

8.1 SendLevel()

This function specifies that events of specified transaction level should be sent to the script.
Currently only frame level events can be sent to verification scripts.

Format: `SendLevel(level)`

Parameters:

level — Can be one of following values:

_PKT:	Packet level (value = 0)
_TRA:	Transaction level (value = 1)
_SPL_TRA:	Split Transaction level (value = 2)
_XFER:	Transfer level (value = 3)
_SCSI:	SCSI Operation level (value = 11)
_PHY_TRA:	PHY Transaction level (value = 15)

Remark:

If no level was specified, events of frame level are sent to the script by default.

Example:

```
SendLevel(_PKT); # Send packet level events.
```

8.2 SendLevelOnly()

This function specifies that ONLY events of a specified transaction level should be sent to the script. Currently only frame level events can be sent to verification scripts.

Format: `SendLevelOnly(level)`

Parameters:

level	Can be one of following values: **_PKT**: Packet level (value = 0) **_TRA**: Transaction level (value = 1) **_SPL_TRA**: Split Transaction level (value = 2) **_XFER**: Transfer level (value = 3) **_SCSI**: SCSI Operation level (value =11) **_PHY_TRA**: PHY Transaction level (value =15)

Example:

```
SendLevelOnly(_PKT); # Send ONLY packet level events.
```

8.3 DontSendLevel()

This function specifies that events of a specified transaction level should NOT be sent to the script. Currently only frame level events can be sent to verification scripts.

Format: `DontSendLevel(level)`

Parameters:

level — Can be one of following values:

- **_PKT**: Packet level (value = 0)
- **_TRA**: Transaction level (value = 1)
- **_SPL_TRA**: Split Transaction level (value = 2)
- **_XFER**: Transfer level (value = 3)
- **_SCSI**: SCSI Operation level (value =11)
- **_PHY_TRA**: PHY Transaction level (value =15)

Example:

```
…
DontSendLevel(_SPL_TRA); # DO NOT send split transaction level events.
```

8.4 SendChannel()

This function specifies that events on the specified channel should be sent to the script.

Format: `SendChannel(channel)`

Parameters:

channel Can be one of channel values defined as in.channel in section 5.1.

Example:

```
SendChannel(_USB2);     # Send events from USB2 channel.
…
SendChannel(_CHANNEL_5); # Send events from channel 5 (USB3_RX).
SendChannel(_CHANNEL_6); # Send events from channel 6 (USB3_TX).
…
SendChannel(5); # Send events from channel 5 (USB3_RX).
SendChannel(6); # Send events from channel 6 (USB3_TX).
…
```

8.5 SendChannelOnly()

This function specifies that ONLY events occuring on a specified channel should be sent to the script.

Format: `SendChannelOnly(channel)`

Parameters:

channel Can be one of channel values defined as in.channel in section 5.1.

Example:

```
SendChannelOnly (_USB2);      # Send ONLY events from USB2 channel.
…
SendChannelOnly (_CHANNEL_5); # Send ONLY events from channel 5 (USB3_RX).
SendChannelOnly (_CHANNEL_6); # Send ONLY events from channel 6 (USB3_TX).
…
SendChannelOnly  (5); # Send ONLY events from channel 5 (USB3_RX).
SendChannel Only (6); # Send ONLY events from channel 6 (USB3_TX).
…
```

8.6 DontSendChannel()

This function specifies that events occuring on a specified channel should NOT be sent to the script.

Format: `DontSendChannel (channel)`

Parameters:

channel Can be one of channel values defined as in.channel in section 5.1.

Example:

```
DontSendChannel(_USB3_TX); # DO NOT send events from USB3 downstream channel.
…
DontSendChannel(_CHANNEL_5);  # DO NOT send events from channel 5 (USB3_RX).
DontSendChannel(_CHANNEL_6);  # DO NOT send events from channel 6 (USB3_TX).
…
DontSendChannel(5); # DO NOT send events from channel 5 (USB3_RX).
DontSendChannel(6); # DO NOT send events from channel 6 (USB3_TX).
…
```

8.7 SendAllChannels()

This function specifies that events occuring on ALL channels should be sent to the script.

Format: `SendAllChannels ()`

Example:

```
…
SendAllChannels (); # Send events from ALL channels.
```

8.8 SendTraceEvent()

This function specifies the events to be sent to a script.

Format: `SendTraceEvent(event)`

Parameters:

event	Can have one of the Trace Event values defined in Chapter 7 "Verification Script Engine Events"

Example:

```
…
SendTraceEvent(_USB2_TOKEN);  # Send USB2 Token packet trace events.
SendTraceEvent(_USB2_BUS_CONDITION); # Send bus condition trace events.
…
SendTraceEvent(_T_SETUP)      # Send Setup Transactions.
```

8.9 DontSendTraceEvent()

This function specifies that the event specified in this function should not be sent to a script.

Format: `DontSendTraceEvent (event)`

Parameters:

event See **SendTraceEvent()** for all possible values.

Example:

```
…
SendTraceEvent(_PACKET); # Send all packets.

…
if(SomeCondition)
{
      DontSendTraceEvent (_USB2_SOF);    # Don't send Start Of Frame packets.
      DontSendTraceEvent (_USB2_SPLIT);  # Don't send Split tokens.
}
```

8.10 SendTraceEventOnly()

This function specifies that ONLY the event specified in this function is sent to the script.

Format: `SendTraceEventOnly(event)`

Parameters:

event	See **SendTraceEvent()** for all possible values

Remark:

This function may be useful, when there are many events to be sent, to send only one kind of event and not send the other ones.

Example:

```
…
SendTraceEvent(_PACKET); # Send all packets.
…
if(SomeCondition)
{
      SendTraceEventOnly (_USB2_TOKEN);
      # Only Token packets are sent.
}
```

8.11 SendAllTraceEvents()

This function specifies that ALL trace events relevant for the selected transaction level are sent to the script.

Format: `SendAllTraceEvents ()`

Example:

```
…
SendAllTraceEvents ();  # Send all types of trace events.
```

8.12 SendDirection()

This function specifies more precise tuning for sending events only for a specified direction of the link.

Format: `SendDirection (direction)`

Parameters:

direction Can have one of the following values:

Primitive Values

Constant	Direction
_IN	Direction is from Host to Device (value = 2)
_OUT	Direction is from Device to Host (value = 3)
_ANY	Any Direction

If this parameter parameter is omitted, trace events of all directions are sent, which is the same as:
`SendTraceEvent(_ANY);`

Example:

```
SendBusState(_IN);    # Send events that were transmitted from Host to Device.
SendBusState();       # Send events in all directions.
SendBusState(_ANY);   # Send events in all directions.
```

8.13 SendUsb2BusConditions()

This function specifies more precise tuning for USB2 Bus Condition events. Only selected bus condition events are sent.

Format: `SendUsb2BusConditions (condition)`

Parameters:

condition — Specifies bus condition.
This parameter can have one of the following values:

BusCondition Constant	Value	Description
_CHIRP_K	0	Device "K" Chirp to notify host of Hi Speed capability
_CHIRP_J	1	Device "J" Chirp to notify host of Hi Speed capability
_FS_K_ON_HS	2	Full Speed "K" signaling on Hi Speed branch
_FS_J_ON_HS	3	Full Speed "J" signaling on Hi Speed branch
_SUSPEND	4	Suspend signaling
_RESUME	5	Resume signaling
_SE1	6	SE1 signaling
_SE0	7	SE0 FS or LS signaling (can be keep-alive for LS)
_VBUS_VOLTAGE_CHANGE	44(0x2C)	Change of Vbus Voltage
_RESET	48(0x30)	Reset signaling
_ANY		All bus condition events

Example:

```
# Send only Resume signaling events.
SendUsb2BusConditions(_RESUME);
…
# Send all bus condition events.
SendUsb2BusConditions (_ANY);
# or
SendUsb2BusConditions ();
```

8.14 SendUsb2TokenPackets()

This function specifies more precise tuning for USB2 Token packets.

Format: `SendUsb2TokenPackets (pid, address, endpoint)`

Parameters:

pid	Specifies that only Token packets with the specified PID value are sent.
	The value **_ANY** means any PID is accepted.
	The possible PID values are
	
	const PID_OUT = 0x87;
	const PID_IN = 0x96;
	const PID_SETUP = 0xB4;
	const PID_SPLIT = 0x1E;
	const PID_PING = 0x2D;
	const PID_EXT = 0x0F;

address	Specifies that only Token packets with the specified Address value are sent. The value **_ANY** means any Address is accepted.
endpoint	Specifies that only Token packets with the specified Endpoint value are sent. The value **_ANY** means any Endpoint is accepted.

Example:

```
# Send any Token packet.
SendUsb2TokenPackets();

# Send all SETUP packets.
SendUsb2TokenPackets (PID_SETUP);

# Send all IN tokens for address 5 endpoint 1.
SendUsb2TokenPackets (PID_IN, 3);

# Send all OUT tokens for address 3.
SendUsb2TokenPackets (PID_OUT, 5, 1);

# Send all token packets for endpoint zero.
SendUsb2TokenPackets (_ANY, _ANY, 0);
```

8.15 SendUsb2DataPackets ()

This function specifies more precise tuning for Data packets.

Format: `SendUsb2DataPackets(pid)`

Parameters:

pid — Specifies that only Data packets with the specified data PID are sent.
The value **_ANY** means any data packet is accepted.
The follofwing PID values apply:

```
const PID_DATA0     = 0xC3;
const PID_DATA1     = 0xD2;
const PID_DATA2     = 0xE1;
const PID_MDATA     = 0xF0;
```

Example:

```
# Send any Data packet.
SendUsb2DataPackets();

# Send only MDATA packets.
SendUsb2DataPackets(PID_MDATA);
```

8.16 SendUsb2HskPackets()

This function specifies more precise tuning for Handshake packets.

Format: `SendUsb2HskPackets(pid)`

Parameters:

pid Specifies that only Data packets with the specified data PID are sent.
The value **_ANY** means any data packet is accepted.
The follofwing PID values apply:

const PID_ACK = 0x4B;
const PID_NAK = 0x5A;
const PID_STALL = 0x78;
const PID_NYET = 0x69;
const PID_ERR = 0x3C;

Example:

```
# Send any handshake.
SendUsb2HskPackets();

# Send only STALL handshakes.
SendUsb2HskPackets (PID_STALL);
```

8.17 SendTransaction()

This function works on the Transaction level and allows specifying more precise tuning for the transaction events that are sent to the script.

Format: `SendTransaction(tra_type, xfer_type, address, endpoint, completion_flag, only_with_errors)`

Parameters:

tra_type	Specifies the type of Transaction event (see transaction event types).
xfer_type	Specifies the type of Transfer this transaction is being part of (see transfer event types).
address	Specifies that only Transaction events with the specified Address value are sent. The value **_ANY** means any Address is accepted.
endpoint	Specifies that only Transaction events with the specified Endpoint value are sent. The value **_ANY** means any Endpoint is accepted.

completion_flag	Specifies that only Transaction events with the specified Handshake PIDs are sent.
	The value **_ANY** means any handshake is accepted.
	The possible values are:
	const PID_ACK = 0x4B;
	const PID_NAK = 0x5A;
	const PID_STALL = 0x78;
	const PID_NYET = 0x69;
	const PID_ERR = 0x3C;

only_with_errors	When set to one, specifies that only Transaction events with errors at the transaction level are sent.

Example:

```
# Send SETUP transactions for any Transfer type and address 0.
SendTransaction( _T_SETUP, _ANY, 0 );

# Send OUT transactions for any transfer type, address 1, endpoint 0.
SendTransaction( _T_OUT, _ANY, 1, 0 );

# Send IN transactions for BULK transfers that were acknoledged by ACK.
SendTransaction( _T_IN, _X_BULK, _ANY, _ANY, PID_ACK );

# Send IN transactions with errors.
SendTransaction( _T_IN, _ANY, _ANY, _ANY, _ANY, 1 );
```

8.18 SendTransfer()

This function works on the Transfer level and allows specifying more precise tuning for the transfer events that are sent to the script.

Format: `SendTransfer(xfer_type, address, endpoint, completed, only_with_errors)`

Parameters:

xfer_type	Specifies the type of Transfer to send to the script (see transfer event types). The value **_ANY** means any transfer type is accepted.
address	Specifies that only Transfer events with the specified Address value are sent. The value **_ANY** means any Address is accepted.
endpoint	Specifies that only Transfer events with the specified Endpoint value are sent. The value **_ANY** means any Endpoint is accepted.
completed	Specifies that only Transfers that are complete (1, default) or incomplete (0) should be sent to the script.
only_with_errors	When set to one, specifies that only Transfer events with errors at the transfer level are sent.

Example:

```
# Send Control Transfers for address 0.
SendTransfer( _X_CONTROL, 0 );


# Send BULK transfers with errors.
SendTransfer( _X_BULK, _ANY, _ANY, 0, 1 );
```

8.19 SendSCSIOperation()

This function works on the SCSI level and allows specifying more precise tuning for the SCSI operation events that are sent to the script.

Format: `SendSCSIOperation(address)`

Parameter:

address	Specifies that only SCSI operation events with the specified Address value are sent. The value **_ANY** means any Address is accepted.

Example:

```
# Send all SCSI operations.
SendSCSIOperation( _ANY );


# Send SCSI operations with address 2.
SendSCSIOperation ( 2 );
```

8.20 SendPHYTransaction()

This function works on the PHY Transaction level and allows specifying more precise tuning for the PHY Transaction events that are sent to the script.

Format: `SendPHYTransaction(phy_tra_type)`

Parameter:

phy_tra_type	Specifies the type of PHY Transaction to send to the script (see PHY Transaction event types). The value **_ANY** means any PHY Transaction type is accepted.

Example:

```
# Send all PHY Transactions.
SendPHYTransaction( _ANY );


# Send PHY Ready Transactions.
SendPHYTransaction ( _P_PHY_READY );
```

8.21 SendPktsWithBadCRC()

This function instructs VSE to send packets with bad CRC to the script. By default, packets with bad CRC are not sent to verification scripts.

- To determine whether a packet has an CRC5 or CRC16 error, check **in.Errors**
  (It is different from 0 if there are some header errors. See the **in.Errors** section for full error bit descriptions.)

Format : **SendPktsWithBadCRC(send_packets_with_bad_crc)**

Parameters:

send_packets_with_bad_crcr	Optional parameter specifies whether frames with bad CRC should be sent to the verification script or not. If it is omitted or set to 1, frames with bad CRC are sent to the script. Otherwise, VSE sends only frames with correct CRC.

Example:

```
SendAllChannels ();      # Send events from ALL channels.
SendAllTaceEvents();     # Send all events.
SendPktsWithBadCRC();    # Send packets with CRC errors.
…
crc16_error_mask = 0x8;  # Bit 3 (if set) indicates CRC16 error.

if( in.Errors & crc16_error_mask)
{
      # Handle CRC16 error.
}
…
SendPktsWithBadCRC ( 0 ); # Send frames with no CRC errors.
```

9 Packet and Script Decoded Fields Retrieving Functions

This group of functions covers VSE generic capability to extract information about data payload fields decoded with CATC Script Language (CSL) at Transfer level and all decoded fields at Packet level for USB3 packets.

9.1 GetDecodedPktField()

Extracts information about a USB3 or Power Delivery packet field and determines how it is shown in the USB Protocol Suite trace view or the "View … Fields" dialog.

Format : `GetDecodedPktField ( fld_name )`

Parameters

fld_name	Name of the field supposedly existing in the current packet

Return Values

If the field name is in the current packet, the return value is the text value of the decoded field and how to display it in the trace view. Otherwise, the return value is the empty string.

Example

```
if( in.TraceEvent == _USB3_LMP_PKT )
{
    # Example of using decoded packet information for LMPs.
    # 'SubType' field
    # String value
    str = FormatEx("\tSubType(str) = '%s'", GetDecodedPktField("SubType"));
    ReportText( str );
}
```

Remark

The field name should be exactly the same as it is in the trace. The field name is case sensitive.
This rule also applies to power delivery packet fields but if the field name contains blank or non-alpha numeric characters, all those characters must be replaced with "_" character
Fields containing ' or ' ' characters are replaced with "Prime" or "DPrime" strings respectively.

9.2 GetHexPktField()

Extracts the raw hexadecimal value of the USB3 packet field.

Format : `GetHexPktField ( fld_name )`

Parameters

fld_name	Name of the field supposedly existing in the current packet

Return Values

If the field name is in the current packet, the function returns the hex value of the decoded field:

- integer value if field length is less than 32 bits
- raw binary value if field length is greater than 32 bits. The raw binary value gives a list of bytes. See the *CSL Manual* for further details about raw binary values.
- null value if the field was not found.

Example

```
# Link Control Word
if( ( in.TraceEvent == _USB3_TP_PKT ) ||
    ( in.TraceEvent == _USB3_DP_PKT ) )
{
    ReportText( "LCW:" );
    val = GetHexPktField ( "Hseq" );
    str = FormatEx( "\tHseq = %d", val );
    ReportText( str );

    val = GetHexPktField ( "Hdepth" );
    str = FormatEx( "\tHDepth = %d", val );
    ReportText( str );

    val = GetHexPktField ( "D1" );
    str = FormatEx( "\tDelayed (D1) = %d", val );
    ReportText( str );

    val = GetHexPktField ( "D2" );
    str = FormatEx( "\tDeferred (D2) = %d\n", val );
    ReportText( str );
}
```

Remark

The field name should be exactly the same as it is in the trace. The field name is case sensitive.

9.3 GetDecodedScriptField()

Extracts information about the script decoded field and determines how it is shown in the USB Protocol Suite trace view or the "View … Fields" dialog.

Format : `GetDecodedScriptField ( fld_name )`

Parameters:

fld_name	Name of the field supposedly existing in the current Transfer

Return Values:

If the field name is in the current Transfer, the return value is the text value of the decoded field and how to display it in the trace. Otherwise, the return value is the empty string.

Example:

```
# Extract the decoded value of wValue field
# for a Control transfer that uses a certain Class
# or Vendor specific decoding.
str  = GetDecodedScriptField ("wValue");

# If the bulk transfer payload decoded by the script decoder
# has a field named 'Code' (i.e., PTP transfers):
str = FormatEx( " Code(str) = '%s' ", GetDecodedScriptField ( "Code" ) );
```

Remarks:

The field name should be exactly the same as it is in the trace. The field name is case sensitive.

This function can be used only at the Transfer level.

9.4 GetHexScriptField()

Extracts the raw hexadecimal value of the script decoded field.

Format : `GetHexScriptField ( fld_name )`

Parameters

fld_name	Name of the field supposedly existing in the current Transfer

Return Values

If the field name is in the current Transfer, the function returns the hex value of the decoded field:

- integer value if field length is less than 32 bits
- raw binary value if field length is greater than 32 bits. The raw binary value gives a list of bytes. See the *CSL Manual* for further details about raw binary values.
- null value if the field was not found.

Example

```
# Extract the hex value of the LBA field for mass storage.
val  = GetHexScriptField ( "Logical Block Addr" );

# Extract the hex value of the SomeBig field.
if( GetHexScriptField ( "Some Big" ) ==  'FE80000000000000' )
     ReportText( "Some Big field = FE80-0000-0000-0000");
```

Remarks

The field name should be exactly the same as it is in the trace. The field name is case sensitive.

This function can be used only at the Transfer level.

9.5 HasPacketError()

Checks if an error is contained in the current packet.

Format : `HasPacketError ( erorr_bit_index )`

Parameters

erorr_bit_index Bit index of the error to check. May be one of the values in table below.

Error type	Bit position	Description
USB4 Errors:		
USB4_SB_FALSE_CONNECT	83	At least 25 us of high-logic is needed for Connect
USB4_SB_FALSE_DISCONNECT	84	At least 14 us of low-logic is needed for Disconnect
USB4_SB_INCOMPLETE_PACKET	85	Incomplete packet
USB4_INVALID_CLSE	86	Invalid CLSE
USB4_CRC_ERROR	87	CRC error
USB4_RSVD_NOT_ZEROED	88	Reserved field not zeroed
USB4_UNEXPECTED_DLE	89	Unexpected DLE symbol
USB4_DATA_SYMBOLS_LEN	90	Invalid command len
USB4_DLE_NOT_DUPLICATED	91	DLE symbol not duplicated
USB4_ETX_MISSING	92	EXT symbol missing
USB4_LEN_FIELD_ERROR	93	LEN field error
USB4_DATA_SIZE_ERROR	94	Incorrect data size
USB4_UNDEFINED_DATA_SIZE_ERROR	95	Undefined data size
USB4_INVALID_LSE_SYMBOL	96	Invalid LSESymbol
USB4_HS_INCOMPLETE_PACKET	97	Incomplete packet
USB4_UNKNOWN_PACKET_TYPE_USS	98	Unidentified Symbol Sequence - USS
USB4_ERROR_PACKET_CONTENT	99	Error packet content
USB4_TS_LANE_NUMBER_ERROR	100	TS Lane Number field contains incorrect value
USB4_TS_LANE_BONDING_ERROR	101	LBT (Lane bonding target) field value does not match LBT2
USB4_UNKNOWN_PACKET_TYPE_IPS	102	Unknown packet - IPS
USB4_ERROR_SCR	103	SCR content error
USB4_ERROR_HEC	104	HEC Error
USB4_ERROR_PAYLOAD_CRC	105	CRC Error
USB4_ERROR_PAYLOAD_ECC	106	ECC Error

USB4_ERROR_CONTROL_PACKET_DATA_SIZE	107	Data Size is either less 0 dw or greater 60 dw
USB4_ERROR_CONTROL_PACKET_ADAPTER_NUM	108	Adapter Num is invalid
USB4_WARNING_NOTIFICATION_EVENT_CODE	109	Unknown Notification Code
USB4_WARNING_NOTIFICATION_PG	110	Notification Packet is not a Hot Plug Acknowledgment
USB4_ERROR_CONTROL_INTERDOMAIN_PACKET_ROUTE_STRING_CM	111	Interdomain CM is invalid
USB4_ERROR_TLP_HEADERS_GAP	112	Invalid gap between HS packets
USB4_ERROR_LEN_AND_DATA_SIZE_NOT_MATCH	113	Length and size values don't match
USB4_ERROR_PAYLOAD_LENGTH	114	Payload length is invalid
USB4_WARNING_INCOMPLETE_SLOS_WAKE_PACKET	115	Incomplete SLOS/WAKE packet
USB4_TUNNELED_PCIE_RSVD_NOT_ZEROED	128	Tunneled PCIe: Reserved field not zeroed
USB4_TUNNELED_PCIE_CRC_ERROR	129	Tunneled PCIe: CRC Error
USB4_TUNNELED_PCIE_INVALID_ENCODING	130	Tunneled PCIe: TLP/DLLP Invalid Encoding
USB4_TUNNELED_PCIE_ERROR_PAYLOAD	131	Tunneled PCIe: TLP Error Payload
USB4_TUNNELED_PCIE_ERROR_LENGTH_FIELD	132	Tunneled PCIe: TLP Error Length
USB4_TUNNELED_PCIE_ERROR_TC_FIELD	133	Tunneled PCIe: TLP TC Error (not 0)
USB4_TUNNELED_PCIE_ERROR_ATTR_FIELD	134	Tunneled PCIe: TLP Attr Error (not 0)
USB4_TUNNELED_PCIE_ERROR_BYTE_ENABLES	135	Tunneled PCIe: TLP Byte Enables Violation
USB4_TUNNELED_PCIE_MEM_ERROR_ADDR_LEN	136	Tunneled PCIe: Memory TLP Address/Length crosses 4K
USB4_TUNNELED_PCIE_MEM_ERROR_WRONG_TYPE	137	Tunneled PCIe: Mem64 TLP used incorrectly
USB4_TUNNELED_PCIE_CFG_ERROR_REGISTER	138	Tunneled PCIe: Cfg TLP Register error
USB4_TUNNELED_PCIE_MSG_ERROR_ROUTING	139	Tunneled PCIe: Msg TLP Invalid Routing
USB4_TUNNELED_PCIE_OS_NOT_START_WITH_BC	140	Tunneled PCIe: OS not starting with BCh
USB4_TUNNELED_PCIE_EIS_PERST_DATA_ERROR	141	Tunneled PCIe: EIS/PERST Payload error (not 0)
USB4_TUNNELED_PCIE_UNKNOWN_PRE_HEADER	142	Tunneled PCIe: TLP/DLLP unknown pre-header
USB4_TUNNELED_PCIE_PRE_HEADER_LEN_ERROR	143	Tunneled PCIe: TLP pre-header LEN field error
USB4_TUNNELED_PCIE_PRE_HEADER_CHK_ERROR	144	Tunneled PCIe: TLP pre-header CHK field error

USB4_TUNNELED_PCIE_RSVD_PDF_ERROR	145	Tunneled PCIe: Reserved PDF is used
USB4_TUNNELED_PCIE_TS_DATA_RATE_ERROR	146	Tunneled PCIe: Training Sequence Data Rate error
USB4_TUNNELED_PCIE_TS_FORMAT_ERROR	147	Tunneled PCIe: Training Sequence Format error
USB4_TUNNELED_PCIE_OS_INVALID_CONTENT	148	Tunneled PCIe: Ordered Set content is Invalid (not TS/EIOS)
USB4_ERROR_MULTIPLE_RETIMER_INDEXES_IN_CL_WAKE	149	Multiple Re-timer Indexes in CL_WAKE (CL_WAKE1, CL_WAKE2)
USB4_ERROR_ZERO_RETIMER_INDEX	150	Zero Re-timer Index in CL_WAKE (CL_WAKE1, CL_WAKE2, CL_WAKE1-2)
USB4_ERROR_DESKEW_SYMBOL_LOCATION	151	Location of De-Skew within symbol is incorrect (each De-Skew should be sent on both Lanes in the same locations within the symbol)
PD_EPR_BATTERY_VARIABLE_NOTSUPPORTED	152	Battery/Variable PDOs are not supported in EPR
USB4_XFER_CAPABILITY_LENGTH_ERROR	153	USB4 Transfer: Capability Length exceeds NextCapPtr value
USB4_ERROR_INVALID_ELT_TYPE	154	Invalid ELTtype
USB4_SB_INVALID_WAKE_DURATION	155	The wake duration is not between 1000 and 9000 ns
USB3_NAKed_DATA_PACKET_ERROR	156	Not Acknowledged DP
USB4_ERROR_INVALID_CELSE	157	Invalid CELSE
USB4_G4_TS_BAD_INDICATION_COMPLEMENT	158	Incorrect bitwise complement of Indication
USB4_G4_TS_BAD_COUNTER	159	Invalid Counter associated with Indication field
USB4_G4_TS_BAD_COUNTER_COMPLEMENT	160	Incorrect bitwise complement of Counter
USB4_G4_TS_BAD_PAYLOAD	161	Error in PRBS11 or PRTS7 part of the TS
USB4_G4_OS_CORRUPTED_SYMBOL	162	Invalid Symbol in OS
USB4_G4_OS_TSNOS_DELAY_EXCEED	163	Delay Symbol exceeds 1000 Symbols
USB4_SB_SHORT_DISCONNECT	164	50 ms of low-logic is needed for SBTX to identify a Disconnect

Return Values

If the current packet contains the required error, the function returns 1 (true), otherwise – 0 (false).

Example

```
    # loop over the range of error indices
    for( err_idx = USB4_FIRST_ERROR_INDEX; err_idx <= USB4_LAST_ERROR_INDEX;
err_idx++ )
    {
        if( HasPacketError( err_idx ) ) # call the function with some error index
        {
            # packet has some error
        }
    }
    # packet has no error from the given range
```

9.6 HasUsb4TraceError()

Checks if USB4 error occurred in a trace file.

Format : `HasUsb4TraceError ( erorr_bit_index )`

Parameters

erorr_bit_index Bit index of the USB4 error to check. May be one of the USB4 values in table from 9.5 HasPacketError() description.

Return Values

If the trace contains the required error, the function returns 1 (true), otherwise – 0 (false).

Example

```
if (HasUsb4TraceError(USB4_ERROR_HEC))
{
    ReportText("HEC error occurred \n");
}
else
{
    ReportText("HEC error has not occurred \n");
}
```

9.7 GetUsb4LegacyTBT3Protocol()

Checks if USB4 trace file was recorded on TBT3 protocol.

Format : `GetUsb4LegacyTBT3Protocol ()`

Return Values

If the trace file was recorded on TBT3 protocol, the function returns 1 (true), otherwise – 0 (false).

Example

```
if (GetUsb4LegacyTBT3Protocol())
{
    ReportText("TBT3 Prorocol \n");
}
else
{
    ReportText("USB4 Protocol \n");
}
```

9.8 GetUsb4HopIDListSize()

Returns the number of HopIDs (for Control packets and Tunneled packets) for the specified channel.

Format : `GetUsb4HopIdListSize( channel )`

Parameters

channel Channel number. May be one of the values from the list of constants for `in.Channel`

Return Values

The number of HopIDs for the specified channel

9.9 GetUsb4HopIDList()

Returns the list of HopIDs (for Control packets and Tunneled packets) for the specified channel.
Note: use “GetUsb4HopIDListSize” function to get the number of HopIDs in the list.

Format : `GetUsb4HopIdList( channel )`

Parameters

channel	Channel number. May be one of the values from the list of constants for `in.Channel`

Return Values

The list of HopIDs for the specified channel

Example

```
hop_id_count = GetUsb4HopIDListSize(_USB4_HS_ANALYZER_LEFT);
hop_ids = GetUsb4HopIDList(_USB4_HS_ANALYZER_LEFT);
for (hop_id_idx = 0; hop_id_idx < hop_id_count - 1; hop_id_idx++)
{
    summary += Format("%d, ", hop_ids[hop_id_idx]);
}
```

9.10 GetUsb4TunneledHopIDListSize()

Returns the number of HopIDs (for Tunneled packets, excluding Control packets) for the specified channel.

Format : `GetTunneledUsb4HopIdListSize( channel )`

Parameters

channel	Channel number. May be one of the values from the list of constants for `in.Channel`

Return Values

The number of HopIDs for the specified channel

9.11 GetUsb4TunneledHopIDList()

Returns the list of HopIDs (for Tunneled packets, excluding Control packets) for the specified channel.
Note: use “GetUsb4HopIDTunneledListSize” function to get the number of HopIDs in the list.

Format : `GetUsb4TunneledHopIdList( channel )`

Parameters

channel Channel number. May be one of the values from the list of constants for `in.Channel`

Return Values

The list of HopIDs for the specified channel

Example

```
hop_id_count = GetUsb4TunneledHopIDListSize(_USB4_HS_ANALYZER_LEFT);
hop_ids = GetUsb4TunneledHopIDList(_USB4_HS_ANALYZER_LEFT);
for (hop_id_idx = 0; hop_id_idx < hop_id_count - 1; hop_id_idx++)
{
    summary += Format("%d, ", hop_ids[hop_id_idx]);
}
```

9.12 GetUsb4LinkType ()

Returns the link type associated with specified port role (for Sideband packets).
Note: use “GetOppositePortRole” function to get the packet port role for opposite side.

Format : `GetUsb4LinkType ( packet_port_role )`

Parameters

packet_port_role Packet port role. A value of `in. PacketPortRole` for Sideband packet.

Return Values

The link type associated with specified packet port role.

Example

```
link_type = GetUsb4LinkType(in.PacketPortRole)
```

9.13 GetOppositePortRole ()

Returns the port role that is opposite to provided one.

Format : `GetOppositePortRole ( packet_port_role )`

Parameters

packet_port_role Packet port role. A value of `in.PacketPortRole` for Sideband packet.

Return Values

The port role for opposite side.

Example

```
port_role = in.PacketPortRole;
if(in.WnR == 1)
{
      # get opposite port role for write request
      port_role = GetOppositePortRole(port_role);
}
link_type = GetUsb4LinkType(port_role);
```

10 Config space exporting functions.

This group of functions covers VSE capability to get Configuration Space Adapter information.

10.1 GetTopologyList()

Return the list of available topologies in opened trace file.

Format : `GetTopologyList()`

Parameters

function has empty parameter list

Return Values

Function returns the list of the topology Id values available in current trace file.

Example

```
topology_list = GetTopologyList();
```

10.2 GetTopologyListSize()

Function returns the size of available topology id list returned by `GetTopologyList()` function.

Format : `GetTopologyListSize()`

Parameters

function has empty parameter list

Return Values

Size of topology ids list obtained by `GetTopologyList()` function..

Example

```
topology_list = GetTopologyList();
topology_list_size = GetTopologyListSize();
str = "Available topology ids: "
for(index = 0; index < topology_list_size; index++)
{
    str += Format("0x%8x ", topology_list[index]);
}
```

10.3 GetAdaptersList()

Function returns the list of adapter numbers obtained for particular topology id.

Format : `GetAdaptersList( topology_id )`

Parameters

topology_id	Topology Id for which the list of the adapter numbers should be obtained

Return Values

List of adapter numbers for particular topology id. Please note that adapter number 0 is not included in returned list. If topology_id is absent in the trace file, the empty adapter number list will be returned.

Example

```
adapter_num_list = GetAdaptersList( topology_id );
```

10.4 GetAdaptersListSize()

Function returns the size of adapter number list returned by `GetAdaptersList()` function.

Format : `GetAdaptersListSize( topology_id )`

Parameters

topology_id	Topology Id for which the size of the adapter number list should be obtained

Return Values

Size of adapter number list obtained by `GetAdaptersList()` function for particular topology_id. If topology_id is absent in trace file, function will return 0.

Example

```
adapters_list = GetAdaptersList(topology_id);
list_size = GetAdaptersListSize(topology_id);
str = "Available adapter numbers: "
for(index = 0; index < list_size; index++)
{
    str += Format("%8d ", adapters_list[index]);
}
```

10.5 GetAdapterCapabilityStartAddresses()

Extract the list of adapter capability start addresses for particular adapter number and topology id.

Format : `GetAdapterCapabilityStartAddresses( topology_id, adapter_num );`

Parameters

topology_id	Topology Id for which requested adapter number belongs
adapter_num	Adapter number for which list of capability start addresses will be returned

Return Values

List of capability start addresses for adapter with number equal to adapter_num and which belongs to the topology identifying by topology_id. If passed topology_id or adapter_num do not exist in the current trace file or trace file does not contain adapter configuration space info, function will return empty list.

Example

```
address_list = GetAdapterCapabilityStartAddresses(topology_id,adapter_num);
```

10.6 GetAdapterCapabilityStartAddressesListSize()

Extract size of adapter capability start address list returned by `GetAdapterCapabilityStartAddresses()` function.

Format : `GetAdapterCapabilityStartAddressesListSize( topology_id, adapter_num );`

Parameters

topology_id	Topology Id for which requested adapter number belongs
adapter_num	Adapter number for which list of capability start addresses will be returned

Return Values

The size of capability start addresses list for adapter_num which belongs to the topology identifying by topology_id. If passed topology_id or adapter_num do not exist in the current trace file or trace file does not contain adapter configuration space info, function will return 0.

Example

```
size = GetAdapterCapabilityStartAddressesListSize(topology_id,adapter_num);
```

10.7 GetAdapterCapabilityID()

Extract capability id from adapter configuration space field identified by topology id, adapter number and field address.

Format : `GetAdapterCapabilityID( topology_id, adapter_num, start_addr );`

Parameters

topology_id	Topology Id for which passed adapter number belongs
adapter_num	Adapter number for which capability id is requested
start_addr	Address of registry field in adapter configuration space

Return Values

Capability id for configuration space of the adapter belonging to particular topology id and which is addressed in configuration space by the start_addr. If passed adapter number and topology id do not exist in trace file, or trace file does not contain configuration space info or start_addr is invalid, this function will return -1.

Example

```
Capability_id = GetAdapterCapabilityID(topology_id,adapter_num, start_addr);
```

10.8 GetAdapterCapabilityName()

Extract string representation of capability from adapter configuration space field which is identified by passed address.

Format : `GetAdapterCapabilityName( topology_id, adapter_num, start_addr );`

Parameters

topology_id	Topology Id for which passed adapter number belongs
adapter_num	Adapter number for which capability name is requested
start_addr	Address of registry field in adapter configuration space

Return Values

Capability name extracted from adapter configuration space registry field addressed by start_addr. If passed topology_id or adapter number are missed in trace file, or trace file does not contain configuration space information, or decoded capability id from registry field addressed in configuration space by start_addr is invalid, function will return “Undefined Capability”.

Example

```
Capability_name = GetAdapterCapabilityName(topology_id,adapter_num,
start_addr);
```

10.9 GetAdapterCapabilityData()

Extract the raw registry field value from particular adapter capability. Requested field value in capability registers is identified by data_offset, capability in adapter configuration space is addressed by start_addr, requested adapter is identified by adapter number and topology id.

Format : `GetAdapterCapabilityData( topology_id, adapter_num, start_addr, data_offset );`

Parameters

topology_id	Topology Id for which passed adapter number belongs
adapter_num	Adapter number for which capability registry field is requested
start_addr	Start address of requested capability in adapter configuration space
data_offset	Address of requested registry field in capability register

Return Values

Raw value of capability registry field addressed by data_offset where the desired adapter capability is given y start_addr in adapter configuration space. If passed adapter number or topology id are absent in trace file, or capability start_addr is invalid, or data_offset is exceed the capability registry size, or requested capability registry field is unknown or has not been initialized, function will return -1.

Example

```
    capability_data = GetAdapterCapabilityData(topology_id,adapter_num, start_addr, data_offset);
```

11 USB4 Tunneled Protocols Assignment

This group of functions covers VSE capability to work with USB4 Tunneled Protocols for USB4 HS Tunneled Packets.

11.1 SetUsb4TunneledProtocol()

This function is depricated, please use SetUsb4TunProtocol () instead.

11.2 SetUsb4TunProtocol()

By default, the assignments in the Decoding Settings are used in VSE, but in case you want to override it with a new Tunneled Protocol, Generation and Link Width to a HopID/Channel, you can use this fucntion.
Please note that this fucntion does not change the assignments in Decoding Settings.

Format : `SetUsb4TunProtocol ( hop_id, channel, tunneled_protocol, gen_type, link_width )`

Parameters

hop_id HopID value of tunneled packet from TLP Header

channel Channel number of current packet. May be one of the values from the list of constatns for `in.Channel`

tunneled_protocol May be one of the values in table below.

Tunneled Protocol	Description
_USB4_TUN_NONE	Remove assignment for specified parameters
_USB4_TUN_USB	Tunneled USB
_USB4_TUN_DP_X1	Tunneled DP Main Link with link width x1
_USB4_TUN_DP_X2	Tunneled DP Main Link with link width x2
_USB4_TUN_DP_X4	Tunneled DP Main Link with link width x4
_USB4_TUN_DPAUX	Tunneled DP Aux
_USB4_TUN_PCIE	Tunneled PCIe
_USB4_TUN_HI	Tunneled Host Interface

gen_type May be one of the values in table below. Currently, this field is only applied to Tunneled USB.

Generation Type	Description
_USB4_TUN_USB3_GEN_X	Tunneled USB3 GenX
_USB4_TUN_USB3_GEN_T	Tunneled USB3 GenT

0	Unknown

link_width May be one of the values in table below. Currently, this field is only applied to Tunneled USB.

Link Width	Description
_LINK_WIDTH_X1	x1
_LINK_WIDTH_X2	x2
_LINK_WIDTH_UNKNOWN	Unknown

Return Values

If channel number is not valid and hence protocol assignment can not be set, the function returns 0. In all other cases it returns 1.

Example

```
    SetUsb4TunProtocol(8, _USB4_HS_ANALYZER_LEFT, _USB4_TUN_USB,
_USB4_TUN_USB3_GEN_T, _LINK_WIDTH_X2);
    SetUsb4TunProtocol(9, _USB4_HS_ANALYZER_LEFT, _USB4_TUN_PCIE, 0,
_LINK_WIDTH_UNKNOWN);
```

11.3 GetUsb4TunneledProtocol()

Get assigned USB4 Tunneled Protocol to specified HopID and channel.

Format : `GetUsb4TunneledProtocol( hop_id, channel, tunneled_protocol )`

Parameters

hop_id HopID value of tunneled packet from TLP Header

channel Channel number of current packet. May be one of the values from the list of constatns for `in.Channel`

Return Values

May return one of the values from the list of constants for USB4 Tunneled Protocols (see SetUsb4TunProtocol()).

Example

```
if(in.TraceEvent == _USB4_HS_TUNNELED_PKT)
{
    protocol = GetUsb4TunneledProtocol(in.Header_HopID, in.Channel);
}
```

12 Timer Functions

This group of functions covers VSE capability to work with timers, internal routines that repeatedly measure a timing interval between different events.

12.1 VSE Time Object

A VSE time object presents time intervals in verification scripts. The verification script time object is a "list" object of two elements.

```
[seconds, nanoseconds]
```

Note: The best way to construct a VSE time object is to use the **Time()** function (see below).

12.2 SetTimer()

Starts a timing calculation from the event at which this function was called.

Format: `SendTimer(timer_id = 0)`

Parameters:

timer_id	Unique timer identifier.

Example:

```
SetTimer();   # Start timing for timer with id = 0.
SetTimer(23); # Start timing for timer with id = 23.
```

Remark:

If this function is called a second time for the same timer ID, it resets the timer and starts timing the calculation again from the point where it was called.

12.3 KillTimer()

Stops a timing calculation for a specific timer and frees related resources.

Format: `KillTimer(timer_id = 0)`

Parameters:

timer_id	Unique timer identifier.

Example:

```
KillTimer();     # Stop timing for timer with id = 0.
KillTimer(23);   # Stop timing for timer with id = 23.
```

12.4 GetTimerTime()

Retrieves a timing interval from the specific timer.

Format: `GetTimerTime (timer_id = 0)`

Parameters:

timer_id	Unique timer identifier.

Return values:

Returns a VSE time object from a timer with id = **timer_id**.

Example:

```
GetTimerTime ();   # Retrieve timing interval for timer with id = 0.
GetTimerTime (23); # Retrieve timing interval for timer with id = 23.
```

Remark:

This function, when called, does not reset the timer.

13 Time Construction Functions

This group of functions is used to construct VSE time objects.

13.1 Time()

Constructs a verification script time object.

Format:
```
Time(nanoseconds)
Time(seconds, nanoseconds)
```

Parameters:

nanoseconds	Number of nanoseconds in specified time
neconds	Number of seconds in specified time

Return values:

First function returns **[0, nanoseconds]**, second one returns **[seconds, nanoseconds]**

Example:

```
Time (50 * 1000);     # Create time object of 50 microseconds.
Time (3, 100);        # Create time object of 3 seconds and 100 nanoseconds.
Time(3 * MICRO_SECS); # Create time object of 3 microseconds.
Time(4 * MILLI_SECS); # Create time object of 4 milliseconds.
```

Note: MICRO_SECS and MILLI_SECS are constants defined in **VS_constants.inc**.

14 Time Calculation Functions

This group of functions covers VSE capability to work with "time", the VSE time objects.

14.1 AddTime()

Adds two VSE time objects.

Format: `AddTime(time1, time2)`

Parameters:

time_1	VSE time object representing the first time interval
time_2	VSE time object representing the second time interval

Return values:

Returns a VSE time object representing a time interval equal to sum of **time_1** and **time_2**

Example:

```
t1 = Time(100);
t2 = Time(2, 200);
t3 = AddTime(t1, t2)  # Returns VSE time object = 2 sec 300 ns.
```

14.2 SubtractTime()

Subtracts two VSE time objects.

Format: `SubtractTime (time1, time2)`

Parameters:

time_1	VSE time object presenting first time interval
time_2	VSE time object presenting second time interval

Return values:

Returns a VSE time object representing a time interval equal to subtraction of **time_1** and **time_2**

Example:

```
t1 = Time(100);
t2 = Time(2, 200);
t3 = SubtractTime (t2, t1)  # Returns VSE time object = 2 sec 100 ns.
```

14.3 MulTimeByInt()

Multiplies a VSE time object by an integer value.

Format: `MulTimeByInt (time, mult)`

Parameters:

time	VSE time object
mult	Multiplier, integer value

Return values:

Returns a VSE time object representing a time interval equal to **time * mul**t

Example:

```
t    = Time(2, 200);
t1 = MulTimeByInt (t, 2)  # Returns VSE time object = 4 sec 400 ns.
```

14.4 DivTimeByInt()

Divides a VSE time object by an integer value.

Format: `DivTimeByInt (time, div)`

Parameters:

time	VSE time object
div	Divider, integer value

Return values:

Returns a VSE time object representing time interval equal to **time / div.**

Example:

```
t   = Time(2, 200);
t1 = DivTimeByInt (t, 2)  # Returns VSE time object = 1 sec 100 ns.
```

15 Time Logical Functions

This group of functions covers VSE capability to compare VSE time objects.

15.1 IsEqualTime()

Verifies that one VSE time object is equal to another VSE time object.

Format: `IsEqualTime (time1, time2)`

Parameters:

time_1	VSE time object representing the first time interval
time_2	VSE time object representing the second time interval

Return values:

Returns 1 if **time_1** is equal to **time_2**,
Returns 0 otherwise.

Example:

```
t1 = Time(100);  t2 = Time(500);
If(IsEqualTime(t1, t2)) DoSomething();
```

15.2 IsLessTime()

Verifies that one VSE time object is less than another VSE time object.

Format: `IsLessTime (time1, time2)`

Parameters:

time_1	VSE time object representing the first time interval
time_2	VSE time object representing the second time interval

Return values:

Returns 1 if **time_1** is less than **time_2**,
Returns 0 otherwise.

Example:

```
t1 = Time(100);  t2 = Time(500);
If(IsLessTime (t1, t2)) DoSomething();
```

15.3 IsGreaterTime()

Verifies that one VSE time object is greater than another VSE time object.

Format: `IsGreaterTime (time1, time2)`

Parameters:

time_1	VSE time object representing the first time interval
time_2	VSE time object representing the second time interval

Return values:

Returns 1 if **time_1** is greater than **time_2**,
Returns 0 otherwise.

Example:

```
t1 = Time(100);  t2 = Time(500);
If(IsGreaterTime (t1, t2)) DoSomething();
```

15.4 IsTimeInInterval()

Verifies that a VSE time object is greater than a minimum VSE time object and less than a maximum VSE time object.

Format: `IsTimeInInterval(min_time, time, max_time)`

Parameters:

min_time	VSE time object representing a minimum interval
time	VSE time object representing a time interval
max_time	VSE time object representing a maximum interval

Return values:

Returns 1 if **min_time** <= **time** <= **max_time**,
Returns 0 otherwise.

Example:

```
t1 = Time(100);
t  = Time(400);
t2 = Time(500);
If(IsTimeInInterval (t1, t, t2)) DoSomething();
```

16 Time Text Functions

This group of functions covers VSE capability to convert VSE time objects into text strings.

16.1 TimeToText()

Converts a VSE time object into text.

Format: `TimeToText (time)`

Parameters:

time	VSE time object

Return values:

Returns a text representation of a VSE time object.

Example:

```
t = Time(100);
ReportText(TimeToText(t)); # See below details for ReportText() function.
```

16.2 TimeToTextNs()

Converts a VSE time object into text in nanoseconds format.

Format: `TimeToTextNs (time)`

Parameters:

time	VSE time object

Return values:

Returns a text representation in nanoseconds format of a VSE time object.

Example:

```
t = Time(100);
ReportText(TimeToTextNs(t)); # See below details for ReportText() function.
```

17 Output Functions

This group of functions covers VSE capability to present information in the output window.

17.1 ReportText()

Outputs text in the output window related to the verification script.

Format: `ReportText (text)`

Parameters:

text Text variable, constant or literal

Example:

```
…
ReportText ("Some text");
…
t = "Some text"
ReportText (t);
…
num_of_packets = in.NumOfPackets;
text = Format("Number of frames = %d", num_of_packets);
ReportText (text);
…
x = 0xAAAA;
y = 0xBBBB;
text = FormatEx("x = 0x%04X, y = 0x%04X", x, y);
ReportText("Text = " + text);
…
```

17.2 EnableOutput()

Enables showing information in the output window and sending COM reporting notifications to COM clients.

Format: `EnableOutput ()`

Example:

```
EnableOutput ();
```

17.3 DisableOutput()

Disables showing information in the output window and sending COM reporting notifications to COM clients.

Format: `DisableOutput ()`

Example:

```
DisableOutput ();
```

18 Information Functions

18.1 GetTraceName()

This function returns the full path and filename of the trace file being processed by VSE.

Format: `GetTraceName(filepath_compatible)`

Parameters:

filepath_compatible	If this parameter is present and not equal to 0, the returned value may be used as part of the filename.

Example:

```
ReportText("Trace path and name = " + GetTraceName());
…
File = OpenFile("C:\\My Files\\" + GetTraceName(1) + "_log.log");

# For trace file with path "D:\Some Traces\Data.usb"
# GetTraceName(1) returns "D_Some Traces_Data.usb"
```

18.2 GetTraceFilePath()

This function returns the file path of the trace file being processed by VSE. It does NOT include the filename itself.

Format: `GetTraceFilePath()`

Parameters:

<none>

Example:

```
ReportText("Trace path = " + GetTraceFilePath());
…
File = OpenFile("GetTraceFilePath() + "log.txt");

# For trace file with path "D:\Some_Traces\Data.usb"
# GetTraceFilePath() returns "D:\Some_Traces\"
```

18.3 GetTraceNameOnly()

This function returns just the filename of the trace file being processed by VSE. No path is included.

Format: `GetTraceNameOnly()`

Parameters:

<none>

Example:

```
ReportText("Trace filename = " + GetTraceNameOnly ());
…
File = OpenFile("D:\" + GetTraceNameOnly ()); // D:\Data.usb

# For trace file with path "D:\Some_Traces\Data.usb"
# GetTraceNameOnly() returns "Data.usb"
```

18.4 GetScriptName()

This function returns the name of the verification script where this function is called. The path is not included when the Verification script is executed by USBSuite UI.

Format: `GetScriptName()`

Example:

```
ReportText("Current script = " + GetScriptName());
```

18.5 GetApplicationFolder()

This function returns the full path to the folder where the application was started.

Format: `GetApplicationFolder()`

Example:

```
ReportText("Application folder = " + GetApplicationFolder ());
```

18.6 GetCurrentTime()

This function returns the string representation of the current system time.

Format: `GetCurrentTime()`

Example:

```
# Yields current time in format "May 18, 2006, 5:49 PM"
ReportText(GetCurrentTime());
```

18.7 GetTraceStartTime()

This function returns the VSE Time object representing the starting time of the events in the trace being processed. Can be used for getting the USB power values within the captured range.

Format: `GetTraceStartTime()`

Example:

```
# Prints out starting time of the trace

start_time = GetTraceStartTime();
ReportText( TimeToText( start_time ) );
```

18.8 GetTraceEndTime()

This function returns the VSE Time object representing the ending time of the events in the trace being processed. Can be used for getting the USB power values within the captured range.

Format: `GetTraceEndTime()`

Example:

```
# Prints out ending time of the trace

end_time = GetTraceEndTime();
ReportText( TimeToText( end_time ) );
```

18.9 GetTriggerPacketNumber ()

This function returns triggered packet number in scenarios where you know the maximum number of packets is < 2 billion. **If the number of packets may be greater than 2 billion, then you must also retrieve the high order 32 bits via the GetTriggerPacketNumberHi() function, and concatenate the bits into the 64-bit value.**

Format: `GetTriggerPacketNumber ()`

Example:

```
      # Prints out triggered packet number in the trace in cases where < 2 billion
packets are acquired.

      trigger_packet_number = GetTriggerPacketNumber ();
      ReportText( " triggered packet number = %d", trigger_packet_number );
```

18.10 GetTriggerPacketNumberHi ()

This function returns the triggered packet number's upper 32 bits in 32 bit format. You will need to concatenate this value with the value returned from GetTriggerPacketNumber() to obtain the full 64-bit packet number. This is not necessary if you know your scenario only captures < 2 billion packets. In that case, you can use the GetTriggerPacketNumber() by itself to get the 32-bit value.

Format: `GetTriggerPacketNumberHi ()`

Example:

```
      # Prints out triggered packet number in the trace

      trigger_packet_number_high_order_32_bits = GetTriggerPacketNumberHi ();
      ReportText( " Upper 32 bits of triggered packet number = %d",
trigger_packet_number_high_order_32_bits );
```

18.11 GetTriggerTime ()

This function returns trigger time stamp. The value is returned in the VSE Time Object format.

Format: `GetTriggerTime ()`

Example:

```
# Prints out trigger time stamp in the trace

trigger_time = GetTriggerTime ();
ReportText( TimeToText(trigger_time ) );
```

18.12 GetLastPacketNumber ()

This function returns the number of the last packet in a trace file.

Format: `GetLastPacketNumber ()`

Example:

```
# Prints out the number of the last packet in the trace

ReportText(FormatEx("Last Packet Number: %llu", GetLastPacketNumber()));
```

18.13 GetLastPacketNumberHi ()

This function returns the last packet in a trace file number's upper 32 bits in 32 bit format. You will need to concatenate this value with the value returned from GetLastPacketNumber() to obtain the full 64-bit packet number. This is not necessary if you know your scenario only captures < 2 billion packets. In that case, you can use the GetLastPacketNumber() by itself to get the 32-bit value.

Format: `GetLastPacketNumberHi()`

Example:

```
      # Prints out the number of the last packet in the trace

      ReportText(FormatEx("Last Packet Number (High): %llu",
GetLastPacketNumberHi()));
```

19 Power Tracker Functions

This functions can be used to retrieve Power Capture parameters by time. Power capture parameters are displayed by the Power Tracker view in the USB Suite application.

Note: MANY power samples can occur before the first packet in a capture, and after the last packet in a capture. The samples are taken from the start of recording until the end at 20 microsecond intervals.

19.1 IsPowerCaptured ()

This function can be used to determine if power parameters were captured for the trace being processed.

Format: `IsPowerCaptured()`

Parameters:

none

Example:

```
if( IsPowerCaptured() )
{
    # retrieve some power parameters by time
    …
}
```

19.2 GetPwrPowerValue()

This function returns the captured value of the Power (in microwatts) at the specified time.

Format: `GetPwrPowerValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Power sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Power with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - Power: %d microwatts",
        TimeToText( start_time ), GetPwrPowerValue( start_time ) ) );

    start_time = AddTime( start_time, step );
}
```

19.3 GetPwrVoltageValue()

This function returns the captured value of the Voltage (in microvolts) at the specified time.

Format: `GetPwrVoltageValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Voltage sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Voltage with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - Voltage: %d microvolts",
        TimeToText( start_time ), GetPwrVoltageValue( start_time ) ) );

    start_time = AddTime( start_time, step );
}
```

19.4 GetPwrCurrentValue()

This function returns the captured value of the Current (in microampers) at the specified time.

Format: `GetPwrCurrentValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Current sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Current with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - Current: %d microampers",
        TimeToText( start_time ), GetPwrCurrentValue( start_time ) ) );

    start_time = AddTime( start_time, step );
}
```

19.5 GetPwrCC1LeftPowerValue()

This function returns the captured value of the Power (in microwatts) for CC1 channel left port at the specified time.

Format: `GetPwrCC1LeftPowerValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Power sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Power with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel left port Power: %d
                          microwatts",
        TimeToText( start_time ), GetPwrCC1LeftPowerValue ( start_time ) ) );

    start_time = AddTime( start_time, step );
}
```

19.6 GetPwrCC1LeftVoltageValue ()

This function returns the captured value of the Voltage (in microvolts) for CC1 channel left port at the specified time.

Format: `GetPwrCC1LeftVoltageValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Voltage sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Voltage with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel left port Voltage: %d
                          microvolts",
        TimeToText( start_time ), GetPwrCC1LeftVoltageValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.7 GetPwrCC1LeftCurrentValue ()

This function returns the captured value of the Current (in microampers) for CC1 channel left port at the specified time.

Format: `GetPwrCC1LeftCurrentValue ( time )`

Parameters:

time	The VSE Time object representing a time at which the Current sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Current with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel left port Current: %d
                          microampers",
        TimeToText( start_time ), GetPwrCC1LeftCurrentValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.8 GetPwrCC2LeftPowerValue()

This function returns the captured value of the Power (in microwatts) for CC2 channel left port at the specified time.

Format: `GetPwrCC2LeftPowerValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Power sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Power with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel left port Power: %d
                        microwatts",
        TimeToText( start_time ), GetPwrCC2LeftPowerValue ( start_time ) ) );

    start_time = AddTime( start_time, step );
```

19.9 GetPwrCC2LeftVoltageValue ()

This function returns the captured value of the Voltage (in microvolts) for CC2 channel left port at the specified time.

Format: `GetPwrCC2LeftVoltageValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Voltage sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Voltage with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel left port Voltage: %d
                          microvolts",
        TimeToText( start_time ), GetPwrCC2LeftVoltageValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.10 GetPwrCC2LeftCurrentValue ()

This function returns the captured value of the Current (in microampers) for CC2 channel left port at the specified time.

Format: `GetPwrCC2LeftCurrentValue ( time )`

Parameters:

time	The VSE Time object representing a time at which the Current sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Current with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel left port Current: %d
                          microampers",
        TimeToText( start_time ), GetPwrCC2LeftCurrentValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.11 GetPwrCC1RightPowerValue()

This function returns the captured value of the Power (in microwatts) for CC1 channel right port at the specified time.

Format: `GetPwrCC1RightPowerValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Power sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Power with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel right port Power: %d
                          microwatts",
        TimeToText( start_time ), GetPwrCC1RightPowerValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.12 GetPwrCC1RightVoltageValue ()

This function returns the captured value of the Voltage (in microvolts) for CC1 channel right port at the specified time.

Format: `GetPwrCC1RightVoltageValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Voltage sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Voltage with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel right port Voltage: %d
                          microvolts",
        TimeToText( start_time ), GetPwrCC1RightVoltageValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.13 GetPwrCC1RightCurrentValue ()

This function returns the captured value of the Current (in microampers) for CC1 channel right port at the specified time.

Format: `GetPwrCC1RightCurrentValue ( time )`

Parameters:

time	The VSE Time object representing a time at which the Current sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Current with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC1 channel right port Current: %d
                          microampers",
        TimeToText( start_time ), GetPwrCC1RightCurrentValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.14 GetPwrCC2RightPowerValue()

This function returns the captured value of the Power (in microwatts) for CC2 channel right port at the specified time.

Format: `GetPwrCC2RightPowerValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Power sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Power with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel right port Power: %d
                          microwatts",
        TimeToText( start_time ), GetPwrCC2RightPowerValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
```

19.15 GetPwrCC2RightVoltageValue ()

This function returns the captured value of the Voltage (in microvolts) for CC2 channel right port at the specified time.

Format: `GetPwrCC2RightVoltageValue( time )`

Parameters:

time	The VSE Time object representing a time at which the Voltage sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Voltage with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel right port Voltage: %d
                          microvolts",
        TimeToText( start_time ), GetPwrCC2RightVoltageValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.16 GetPwrCC2RightCurrentValue ()

This function returns the captured value of the Current (in microampers) for CC2 channel right port at the specified time.

Format: `GetPwrCC2RightCurrentValue ( time )`

Parameters:

time	The VSE Time object representing a time at which the Current sample value should be returned. Null value will be returned if power wasn't caputured or time is out of range.

Example:

```
start_time = GetTraceStartTime();
step = Time( 1, 0 ); # 1 second

# Prints out 5 samples of Current with 1 second step from the start of trace
for( i=0; i<5; i++ )
{
    ReportText( FormatEx( "At time %s - CC2 channel right port Current: %d
                          microampers",
        TimeToText( start_time ), GetPwrCC2RightCurrentValue ( start_time ) )
);

    start_time = AddTime( start_time, step );
}
```

19.17 GetPwrMinValue ()

This function returns the minimum value of the specified parameter on the specified pin and in the specified time range.

Format: `GetPwrMinValue ( type_c_pin, power_param, from_time, to_time )`

Parameters:

type_c_pin

Type-C Pins Constants	Description
_TYPE_C_PIN_VBUS	Vbus pin
_TYPE_C_PIN_CC1	CC1 pin
_TYPE_C_PIN_CC2	CC2 pin
_TYPE_C_PIN_CC1_LEFT	Analyzer left CC1 pin
_TYPE_C_PIN_CC1_RIGHT	Analyzer right CC1 pin
_TYPE_C_PIN_CC2_LEFT	Analyzer left CC2 pin
_TYPE_C_PIN_CC2_RIGHT	Analyzer right CC2 pin

power_param

Power Parameter Constants	Description
_PWR_VOLTAGE	Voltage
_PWR_CURRENT	Current
_PWR_POWER	Power

from_time The start time of the time range.

to_time The end time of the time range.

Example:

```
start_time = GetTraceStartTime();
end_time = GetTraceEndTime();

min_voltage = GetPwrMinValue( _TYPE_C_PIN_VBUS, _PWR_VOLTAGE, start_time,
end_time );

ReportText( FormatEx( "Min Voltage On Vbus : %d", min_voltage ) );
```

19.18 GetPwrMaxValue ()

This function returns the maximum value of the specified parameter on the specified pin and in the specified time range.

Format: `GetPwrMaxValue ( type_c_pin, power_param, from_time, to_time )`

Parameters:

type_c_pin

Type-C Pins Constants	Description
_TYPE_C_PIN_VBUS	Vbus pin
_TYPE_C_PIN_CC1	CC1 pin
_TYPE_C_PIN_CC2	CC2 pin
_TYPE_C_PIN_CC1_LEFT	Analyzer left CC1 pin
_TYPE_C_PIN_CC1_RIGHT	Analyzer right CC1 pin
_TYPE_C_PIN_CC2_LEFT	Analyzer left CC2 pin
_TYPE_C_PIN_CC2_RIGHT	Analyzer right CC2 pin

power_param

Power Parameter Constants	Description
_PWR_VOLTAGE	Voltage
_PWR_CURRENT	Current
_PWR_POWER	Power

from_time The start time of the time range.

to_time The end time of the time range.

Example:

```
start_time = GetTraceStartTime();
end_time = GetTraceEndTime();

max_current = GetPwrMaxValue( _TYPE_C_PIN_VBUS, _PWR_CURRENT, start_time,
end_time );

ReportText( FormatEx( "Max Current On Vbus : %d", max_current ) );
```

20 Navigation Functions

20.1 GotoEvent()

This function forces the application to jump to some trace event and shows it in the main trace view.

Format:
```
GotoEvent(level, index)
GotoEvent()
```

Parameters:

level	Transaction level of the event to jump
index	Transaction index of the event to jump

Remarks:

If no parameters were specified, the application jumps to the current event being processed by VSE.

If wrong parameters were specified (for example, index exceeding the maximal index for the specified transaction level), the function does nothing and an error message is sent to the output window.

Example:

```
 …
 if(Something == interesting) GotoEvent(); # Go to the current event.
 …
 if(SomeCondition)
 {
   interesting_level  = in.Level;
   interesting_index = in.Index;
 }
…
OnFinishScript()
{
 …
 # Go to the interesting event.
 GotoEvent(interesting_level, interesting_index);
}
```

20.2 SetMarker()

This function sets a marker for some trace event.

Format: `SetMarker(marker_text)`
`SetMarker(marker_text, level, index)`

Parameters:

marker_text	Text of the marker
level	Transaction level of the event to jump
index	Transaction index of the event to jump

Remarks:

If no parameters were specified, other than **marker_text,** the application sets the marker to the current event being processed by VSE.

If wrong parameters were specified (for example, index exceeding the maximal index for the specified transaction level), the function does nothing and an error message is sent to the output window.

Example:

```
…
# Set marker to the current event.
if(Something == interesting) SetMarker("!!! Something cool !!!");
…
if(SomeCondition)
{
  interesting_level = in.Level;
  interesting_index = in.Index;
}
…
OnFinishScript()
{
…
# Set marker to the interesting event.
SetMarker("  !!! Cool Marker !!! ", interesting_level, interesting_index);

# Go to the interesting event.
GotoEvent(interesting_level, interesting_index);
}
```

20.3 MarkerContainsText()

This function checks if the current trace event contains the specified text.

Format: `MarkerContainsText()`
`MarkerContainsText( text )`

Parameters:

text	Case-insensitive text for seaching

Remarks:

If no parameter is specified, the function will check whether trace event has any marker or not.

Example:

```
…
# Check if the current event has marker.
if( MarkerContainsText() )
{
      ReportText(" There is a marker!");

      # Check if the current trace event marker contains "Triggered" phrase.
      If( MarkerContainsText("Triggered") )
      {
            ReportText(" Current marker already has <Triggered> text!");
      }
}
…
```

21 File Functions

This group of functions covers VSE capabilities to work with external files.

21.1 OpenFile()

This function opens a file for writing.

Format: `OpenFile(file_path, append, mode)`

Parameters:

file_path	Full path to the file to open. (For '\' use '\\')
append	This parameter (if present and not equal to 0) specifies that VSE should append the following write operations to the contents of the file; otherwise, the contents of the file are cleared.
mode	This parameter specifies the open mode (text or binary) for the file. If omitted, text mode is used. It can be one of the following values: **_FO_BINARY** (= 0): Opens file in binary mode **_FO_TEXT** (= 1): Opens file in text mode (by default)

Return Values:

The "handle" to the file to be used in other file functions.

Example:

```
…
set file_handle = 0;
…
file_handle = OpenFile("D:\\Log.txt");  # Opens file in text mode.
                                        # The previous contents
                                        # are erased.
…
WriteString(file_handle, "Some Text1"); # Write text string to file.
WriteString(file_handle, "Some Text2"); # Write text string to file.
…
CloseFile(file_handle);                 # Closes the file.
…
# Opens file in text mode.
# The following file operations append to the contents of the file.
file_handle = OpenFile(GetApplicationFolder() + "Log.txt", _APPEND);

# Opens file in binary mode. The previous contents are erased.
file_handle = OpenFile("D:\\data.bin", 0, _FO_BINARY);
…
Write(file_handle, 0xAABBCCDD); # Writes integer value into the binary file.
CloseFile(file_handle);         # Closes the binary file.
```

21.2 CloseFile()

This function closes an opened file.

Format: `CloseFile(file_handle)`

Parameters:

file_handle	File "handle"

Example:

```
…
set file_handle = 0;
…
file_handle = OpenFile("D:\\Log.txt"); # Opens file.
                                       # The previous contents are erased.
…
WriteString(file_handle, "Some Text1"); # Write text string to file.
WriteString(file_handle, "Some Text2"); # Write text string to file.
…
CloseFile(file_handle); # Closes file.
…
```

21.3 WriteString()

This function writes a text string to the file.

Format: `WriteString(file_handle, text_string)`

Parameters:

file_handle	File "handle"
text_string	Text string

Remarks:

If the **file_handle** parameter refers to a text file, then a trailing **End-Of-Line** symbol is added to the text. In case of binary files, the **End-Of-Line** symbol is not added.

Example:

```
…
set file_handle = 0;
…
file_handle = OpenFile("D:\\Log.txt"); # Opens text file.
                                       # The previous contents are erased.
…
WriteString(file_handle, "Some Text1"); # Write text string to file.
WriteString(file_handle, "Some Text2"); # Write text string to file.
…
CloseFile(file_handle);                 # Close the file.
…
```

21.4 Write()

This function writes data to the file.

Format: ***Write(file_handle, value, num_of_bytes)***

Parameters:

file_handle	"handle" to the file previously opened by **OpenFile()**
value	Data to write
num_of_bytes	Optional parameter specifying how many bytes to take from the **value** parameter (if omitted, length is calculated automatically based on the value type)

Remarks:

This function is primarily for binary files. It can be used for text files only if the **value** parameter is a string of text. In that case, it is equivalent to the **WriteString()** function and the **num_of_bytes** parameter is ignored.

Example:

```
…
set BinFile = 0;
…
BinFile = OpenFile("C:\\data.bin", 0, _FO_BINARY);
…
# Write a string to the binary file.
Write(BinFile, "All we need is love!!!");

# Write a substring ("All") to the binary file.
Write(BinFile, "All we need is love!!!", 3);

val = 0xBEEF;
Write(BinFile, val);    # Writes integer = (EF BE 00 00) to the binary file.
Write(BinFile, val, 2); # Writes WORD = (EF BE) to the binary file.

# Write a byte chain to the binary file.
Write(BinFile, 'AABBCCDDEEFF12345678');

# Write a list of values to the binary file.
Write(BinFile, [ 0xAA, "USB", 12, 0xBEEF ]);
…
CloseFile(BinFile); # Closes the binary file.
…
```

21.5 ShowInBrowser()

This function opens a file in the Windows Explorer. If the extension of the file has the application registered to open files with such extensions, it is launched. For instance, if Internet Explorer is registered to open files with extension **.htm** and the file handle passed to **ShowInBrowser()** function belongs to a file with such an extension, then this file is opened in the Internet Explorer.

Format: `ShowInBrowser (file_handle)`

Parameters:

file_handle	File "handle"

Example:

```
…
set html_file = 0;
…
html_file = OpenFile("D:\\Log.htm");
…
WriteString(html_file, "<html><head><title>LOG</title></head>");
WriteString(html_file, "<body>");
…
WriteString(html_file, "</body></html>");
ShowInBrowser(html_file);     # Opens the file in Internet Explorer.
CloseFile(html_file);
…
```

22 Trace File Exporting Functions

This group of functions covers VSE capabilities to export data into trace files.

22.1 CreateTraceFile()

This function creates a trace file.

Format: `CreateTraceFile( file_path )`

Parameters:

file_path	Full path to the trace file to be created. (For **'\'** use **'\\'**)

Return Values:

The "handle" to the trace file to be used in other trace-file-related functions.

Remarks:

If **file_path** contains just a file name (such as **test.usb**), the trace file is created in the application folder.

Example:

```
…
set trace_handle = 0;
…
# Create a trace file in "C:\\data_output.usb".
trace_handle = CreateTraceFile("C:\\data_output.usb");
…
# Add the current trace event (frame) to the trace file.
AddEventToTraceFile(trace_handle);
…
CloseTraceFile(trace_handle);          # Close the file.
…
```

22.2 CloseTraceFile()

This function closes an open trace file. All attempts to add trace data to a closed trace file are ignored.

Format: `CloseTraceFile( tracefile_handle )`

Parameters:

tracefile_handle	Trace File "handle" returned by the CreateTraceFile() function.

Example:

```
…
set trace_handle = 0;
…
# Create a trace file in "C:\\data_output.usb".
trace_handle = CreateTraceFile("C:\\data_output.usb");
…
# Add the current trace event (frame) to the trace file.
AddEventToTraceFile(trace_handle);
…
CloseTraceFile(trace_handle);        # Close the file.
…
```

22.3 AddEventToTraceFile()

This function adds the current trace event (such as frames) to the trace file.

Format: `AddEventToTraceFile ( tracefile_handle )`

Parameters:

tracefile_handle	Trace File "handle" returned by the CreateTraceFile() function.

Remarks:

If this function is called multiple times for the same event, only the first call adds an event to the trace file. All the other calls are ignored.

Example:

```
…
set trace_handle = 0;
…
# Create a trace file in "C:\\data_output.usb".
trace_handle = CreateTraceFile("C:\\data_output.usb");

…
# Add the current trace event (frame) to the trace file.
AddEventToTraceFile(trace_handle);
…
CloseTraceFile(trace_handle);            # Close the file.
…
```

22.4 OpenTraceFile()

This function opens a trace file in the application.

Format: `OpenTraceFile (tracefile_path)`

Parameters:

tracefile_path	Path to a trace file to open.

Remarks:

If **tracefile_path** contains just a file name (such as **test.usb**), VSE looks for a trace file with such a name in the application folder.

Example:

```
…
set trace_handle = 0;
…
# Create a trace file in "C:\\data_output.usb".
trace_handle = CreateTraceFile("C:\\data_output.usb");
…
# Add the current trace event (frame) to the trace file.
AddEventToTraceFile(trace_handle);
…
CloseTraceFile(trace_handle);          # Close the trace file.
…
# Instruct the application to open the trace file "C:\\data_output.usb".
OpenTraceFile("C:\\data_output.usb");
```

23 COM/Automation Communication Functions

This group of functions covers VSE capabilities to communicate with COM/Automation™ clients connected to the application. (See the *Automation Manual* for details about how to connect to the application and VSE.)

23.1 NotifyClient()

This function sends information to COM/Automation™ client applications in custom format. The client application receives a VARIANT object, which it is supposed to parse.

Format: `NotifyClient(event_id, param_list)`

Parameters:

event_id	Integer parameter representing event identifier
param_list	List of parameters to be sent to the client application. Each parameter might be an integer, string, raw data (byte chain), or list. Because the list itself may contain integers, strings, raw data, or other lists, it is possible to send very complicated messages. (Lists should be treated as arrays of VARIANTs.)

Example:

```
...
if(SomeCondition())
{
      NotifyClient(2, [ in.Index, "USB2 CHANNEL", "USB2 packet", in.Pid,
                      in. PayloadLength, TimeToText(in.Time)]);
}
...
# Send two parameters to client applications:
# 2 (integer),
# [ in.Index, "USB2 CHANNEL", "USB2 packet", in.Pid,
#   in. PayloadLength, TimeToText(in.Time)] (list)
```

Remark:

See an example of handling this notification by client applications and parsing code in the *Automation Manual*.

24 User Input Functions

24.1 MsgBox()

Displays a message in a dialog box, waits for the user to click a button, and returns an Integer indicating which button the user clicked.

Format: `MsgBox(prompt, type, title)`

Parameters:

prompt	Required. String expression displayed as the message in the dialog box.
type	Optional. Numeric expression that is the sum of values specifying the number and type of buttons to display, the icon style to use, the identity of the default button, and the modality of the message box. If omitted, the default value for buttons is **_MB_OK**. (See the list of possible values in the table below.)
title	Optional. String expression displayed in the title bar of the dialog box. If title is omited, the script name is placed in the title bar.

The **type** argument values are:

Constant	Description
_MB_OKONLY	Display **OK** button only (by Default).
_MB_OKCANCEL	Display **OK** and **Cancel** buttons.
_MB_RETRYCANCEL	Display **Retry** and **Cancel** buttons.
_MB_YESNO	Display **Yes** and **No** buttons.
_MB_YESNOCANCEL	Display **Yes**, **No**, and **Cancel** buttons.
_MB_ABORTRETRYIGNORE	Display **Abort**, **Retry**, and **Ignore** buttons.
_MB_EXCLAMATION	Display **Warning Message** icon.
_MB_INFORMATION	Display **Information Message** icon.
_MB_QUESTION	Display **Warning Query** icon.
_MB_STOP	Display **Critical Message** icon.
_MB_DEFBUTTON1	First button is default.
_MB_DEFBUTTON2	Second button is default.
_MB_DEFBUTTON3	Third button is default.
_MB_DEFBUTTON4	Fourth button is default.

Return Values:

This function returns an integer value indicating which button was clicked.

Constant	Description
_MB_OK	**OK** button was clicked.
_MB_CANCEL	**Cancel** button was clicked.
_MB_YES	**Yes** button was clicked.
_MB_NO	**No** button was clicked.
_MB_RETRY	**Retry** button was clicked.
_MB_IGNORE	**Ignore** button was clicked.
_MB_ABORT	**Abort** button was clicked.

Remark:

This function works only for VS Engines controlled via a GUI. For VSEs controlled by COM/Automation™ clients, it does nothing.

This function "locks" the application, which means that there is no access to other application features until the dialog box is closed. To prevent too many MsgBox calls (in case a script is not written correctly), VSE keeps track of all function calls demanding user interaction and doesn't show dialog boxes if some customizable limit is exceeded (returns **_MB_OK** in this case).

Example:

```
…
 if(Something)
 {
 …
 str = "Something happened!!!\nShould we continue?"
 result = MsgBox(str ,
      _MB_YESNOCANCEL | _MB_EXCLAMATION,
      "Some Title");

 if(result != _MB_YES)
      ScriptDone();
      # Go on…
 }
```

24.2 InputBox()

Displays a prompt in a dialog box, waits for the user to input text or click a button, and returns a CSL list object or string containing the contents of the text box.

Format: `InputBox(prompt, title, default_text, return_type)`

Parameters:

prompt	Required. String expression displayed as the message in the dialog box.
title	Optional. String expression displayed in the title bar of the dialog box. If title is omitted, the script name is placed in the title bar.
default_text	Optional. String expression displayed in the text box as the default response if no other input is provided. If default_text is omited, the text box is displayed empty.
return_type	Optional. It specifies the contents of the return object.

The **return_type** argument values are:

Constant	Value	Description
_IB_LIST	0	CSL list object is returned (by Default).
_IB_STRING	1	String input as it was typed in the text box

Return Values:

Depending on the **return_type** argument, this function returns either a CSL list object or the text typed in the text box as it is.

If **return_type = _IB_LIST** (by default), the text in the text box is considered to be a set of list items divided by **','** (only hexadecimal, decimal and string items are currently supported).

For example, the text:

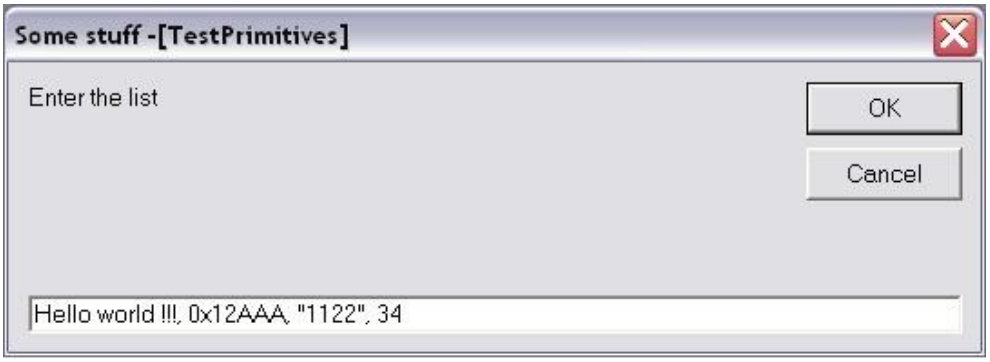
```
Hello world !!!, 12, Something, 0xAA, 10, "1221"
```

produces a CSL list object (5 items):

```
list = [ "Hello world !!!", 12, "Something", 0xAA, 10, "1221" ];

list [0] = "Hello world !!!"
list [1] = 12
list [2] = "Something"
list [3] = 0xAA
list [4] = 10
list [5] = "1212"
```

Note: Although the dialog box input text parser tries to determine a type of list item automatically, text enclosed in quote signs " " is always considered to be a string.



Remark:

This function works only for VS Engines controlled via a GUI. For VSEs controlled by COM/Automation™ clients, it does nothing.

This function "locks" the application, which means that there is no access to other application features until the dialog box is closed. To prevent too many InputBox calls (in case a script is not written correctly), VSE keeps track of all function calls demanding user interaction and does not show dialog boxes if some customizable limit is exceeded. (In that case, it returns a null object.)

Example:

```
…
 if(Something)
 {
 …
 v = InputBox("Enter the list", "Some stuff",
           "Hello world !!!, 0x12AAA, Some, 34");
 ReportText (FormatEx("input = %s, 0x%X, %s, %d", v[0],v[1],v[2],v[3]));
 …    # Go on…

 str = InputBox("Enter the string", "Some stuff",
            "<your string>", _IB_STRING);
 ReportText(str);
 }
```

24.3 GetUserDlgLimit()

This function returns the current limit of user dialogs allowed in the verification script. If the script reaches this limit, no user dialogs are shown and the script does not stop. By default, this limit is set to 20.

Format: `GetUserDlgLimit()`

Example:

```
…
result = MsgBox(Format("UserDlgLimit = %d", GetUserDlgLimit()),
                     _MB_OKCANCEL | _MB_EXCLAMATION, "Some Title !!!");

SetUserDlgLimit(2);   # Set the limit to 2.
…
```

24.4 SetUserDlgLimit()

This function sets the current limit of user dialogs allowed in the verification script. If the script reaches this limit, no user dialogs are shown and the script does not stop. By default, this limit is set to 20.

Format: `SetUserDlgLimit()`

Example:

```
…
result = MsgBox(Format("UserDlgLimit = %d", GetUserDlgLimit()),
                      _MB_OKCANCEL | _MB_EXCLAMATION, "Some Title !!!");

SetUserDlgLimit(2);    # Set the limit to 2.
…
```

25 String Manipulation/Formating Functions

25.1 FormatEx()

Writes formatted data to a string. **Format** is used to control the way that arguments print out. The format string may contain conversion specifications that affect the way in which the arguments in the value string are returned. Format conversion characters, flag characters, and field width modifiers are used to define the conversion specifications.

Format: `FormatEx (format_string, argument_list)`

Parameters:

format_string	Format-control string
argument_list	Optional list of arguments to fill in the format string

Return Values:

Formatted string

Format conversion characters:

Code	Type	Output
c	Integer	Character
d	Integer	Signed decimal integer
i	Integer	Signed decimal integer
o	Integer	Unsigned octal integer
u	Integer	Unsigned decimal integer
x	Integer	Unsigned hexadecimal integer, using "abcdef."
X	Integer	Unsigned hexadecimal integer, using "ABCDEF."
s	String	String

Remarks:

A conversion specification begins with a percent sign (**%**) and ends with a conversion character. The following optional items can be included, in order, between the **%** and the conversion character to further control argument formatting.

Flag characters are used to further specify the formatting. There are five flag characters:

- A **minus sign** (-) causes an argument to be left-aligned in its field. Without the minus sign, the default position of the argument is right-aligned.
- A **plus sign** inserts a plus sign (+) before a positive signed integer. This only works with the conversion characters **d** and **i**.
- A **space** inserts a space before a positive signed integer. This only works with the conversion characters **d** and **i**. If both a space and a plus sign are used, the space flag is ignored.
- A **hash mark** (#) prepends a 0 to an octal number when used with the conversion character **o**. If # is used with **x** or **X**, it prepends 0x or 0X to a hexadecimal number.
- A **zero** (0) pads the field with zeros, instead of with spaces.

Field width specification is a positive integer that defines the field width, in spaces, of the converted argument. If the number of characters in the argument is smaller than the field width, then the field is padded with spaces. If the argument has more characters than the field width has spaces, then the field expands to accommodate the argument.

Example:

```
str = "String";
i = 12;
hex_i = 0xAABBCCDD;
…
formatted_str = FormatEx("%s, %d, 0x%08X", str, i, hex_i);

# formatted_str = "String, 12, 0xAABBCCDD"
```

26 Miscellaneous Functions

26.1 ScriptForDisplayOnly()

Specifies that the script is designed for displaying information only and that its author doesn't care about the verification script result. Such a script has a result **<DONE>** after execution.

Format: `ScriptForDisplayOnly ()`

Example:

```
ScriptForDisplayOnly();
```

26.2 Sleep()

Asks VSE not to send any events to a script until the timestamp of the next event is greater than the timestamp of the current event plus a sleeping time.

Format: `Sleep(time)`

Parameters:

time	VSE time object specifying sleep time

Example:

```
# Don't send any event that occurred during 1 ms from the current event.
Sleep (Time(1000));
```

26.3 ConvertToHTML()

This function replaces spaces with "** **" and carriage return symbols with "**
**" in a text string.

Format: **`ConvertToHTML(`*`text_string`*`)`**

Parameters:

text_string	Text string

Example:

```
str = "Hello world !!!\n";
str += "How are you today?";

html_str = ConvertToHTML (str);
# html_string = "Hello&nbspworld !!!<br>How&nbspare&nbspyou&nbsptoday?"
```

Note: Some other useful miscellaneous functions can be found in the file: **VSTools.inc.**

26.4 **Pause()**

Pauses script running. Later, script execution can be resumed or cancelled.

Format: `Pause()`

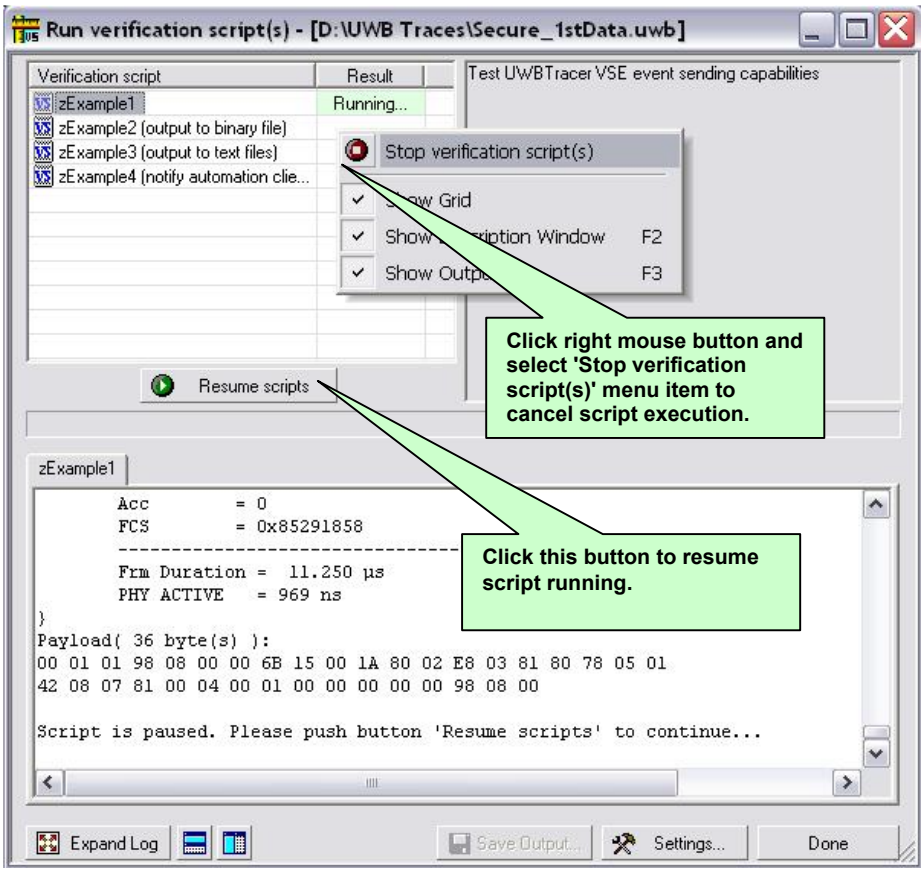
Example:

```
…
If(Something_Interesting())
{
      GotoEvent();     # Jump to the trace view.
      Pause();         # Pause script execution.
}
…
```

Remark:

This function works only for a VS Engine controlled via a GUI. For VSEs controlled by COM/Automation™ clients, it does nothing.

When script execution is paused, the Run Verification Script window looks like:



27 The Important VSE Script Files

The VSE working files are located in the ..**\Scripts\VFScripts** subfolder of the main application folder. The current version of VSE includes following files:

File	Description
VSTools.inc	Main VSE file containing definitions of some useful VSE script functions provided by Teledyne LeCroy (must be included in every VS) **NOTE**: The files VS_constants.inc and VS_Primitives.inc are included.
VS_constants.inc	File containing definitions of some important VSE global constants
VSTemplate.vs_	Template file for new verification scripts
VSUser_globals.inc	File of user global variables and constants definitions (put the definitions of constants, variables, and functions used in many scripts here)

28 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.