Protocol Solutions Group
3385 Scott Blvd., Santa Clara, CA 95054
Tel: +1/408.727.6600
Fax: +1/408.727.6622

Verification Script Engine

for

Teledyne LeCroy PCIe Protocol Suite™

Reference Manual

For PCIe Protocol Suite software version 7.34

Generated: 7/8/2015 10:35 AM

Document Disclaimer

Trademarks and Servicemarks

Copyright

Table of Contents

1 Introduction

This document contains a description of the Teledyne LeCroy Verification Script Engine (VSE), a utility in the PCIe Protocol Suite™ software that allows users to perform custom analyses of PCI Express™ (PE) traffic, recorded using the new generation of PCI Express protocol analyzers.

VSE allows users to ask the PCIe Protocol Suite application to send some desired “events” (currently defined as packets, link transactions, split transactions, AHCI, ATA, NVM transactions or commands) from a PE trace to a verification script written using the CATC script language. This script then evaluates the sequence of events (timing, data or both) in accordance with user-defined conditions and performs post-processing tasks; such as exporting key information to external text-based files or sending special Automation/COM notifications to user client applications.

VSE was designed to allow users to easily retrieve information about any field in a PE packet header or link/split/NVM/AHCI/ATA transaction or command, and to make complex timing calculations between different events in a pre-recorded trace. It also allows filtering-in or filtering-out of data with dynamically changing filtering conditions, porting of information to a special output window, saving of data to text files, and sending of data to COM clients connected to a PCIe Protocol Suite application.

2 Verification Script Structure

Writing a verification script is easy, as long as you follow a few rules and have some understanding of how the PCIe Protocol Suite™ application interacts with running scripts.

The main script file that contains the text of the verification script should have extension **.pevs**, and be located in the subfolder **..\Scripts\VFScripts** of the main PCIe Protocol Suite folder. Some other files might be included in the main script file using directive **%include**. (see the Teledyne LeCroy PCIe Protocol Suite File Based Decoding user manual for details).

The following schema presents a common structure of a verification script (this is similar to the content of the script template [**VSTemplate.pev_**] which is included with VSE):

```
#   VS1.pevs
#
#   Verification script
#
#   Brief Description:
#   Performs specific verification
#

###############################################################################################
   #  Module info
###############################################################################################
#  Filling of this block is necessary for proper verification script operation...
##############################################################################################
set ModuleType  = "Verification Script";            # Should be set for all verification scripts
set OutputType  = "VS";                             # Should be set for all verification scripts that
                                                    # output only Report string and Result.
set InputType   = "VS";

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()
######################################################################################
#
#        It is a 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, all packet level events from all channels
  #  are passed to the script.
  #
  #  For details - how to specify what kind of events should be passed to the script
  #  please see the topic 'sending functions'.
  #
  #  OPTIONAL.
  ###################################################################################

   g_MyGlobalVariable   = 0;
    # Uncomment the line below - if you want to disable output from
    # ReportText()-functions.
    #
    # DisableOutput();
}
###################################################################################
#   ProcessEvent()
###################################################################################
#
###################################################################################
# It is a main script function called by the application when the next waited event
# occured in the evaluated trace.
#
# !!! REQUIRED !!! - MUST BE IMPLEMENTED IN VERIFICATION SCRIPT
###################################################################################

ProcessEvent()
{

    #  Write the body of this function depending upon your needs.
    #  It might require branching on event type:
    # select {
    #   in. TraceEvent == … : …
    #   in. TraceEvent == … : …
    #  …
    # }
    return Complete();
}

###################################################################################
#   OnFinishScript()
###################################################################################
#
###################################################################################
# It is a main script function called by the application when the script 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 == "My Arg" ) return 1;
       return 0;
}
```

3 Interaction between PCIe Protocol Suite and a verification script

When a user runs a script against a pre-recorded trace, the following sequence occurs:

1. Prior to sending information to the script's main processing function **ProcessEvent()**, VSE looks for the function **OnStartScript()** and calls it if it is found. In this function, setup actions are defined, such as specifying the kind of trace events that should be passed to the script and setting up initial values for script-specific global variables.

2. Next, the VSE parses the recorded trace to verify that the current packet or other event meets specific criteria – if it does, VSE calls the script's main processing function **ProcessEvent()**, placing information about the current event in the script's input context variables.
   (Please refer to the topic **Input context variables** later in this document for a full description of verification script input context variables)

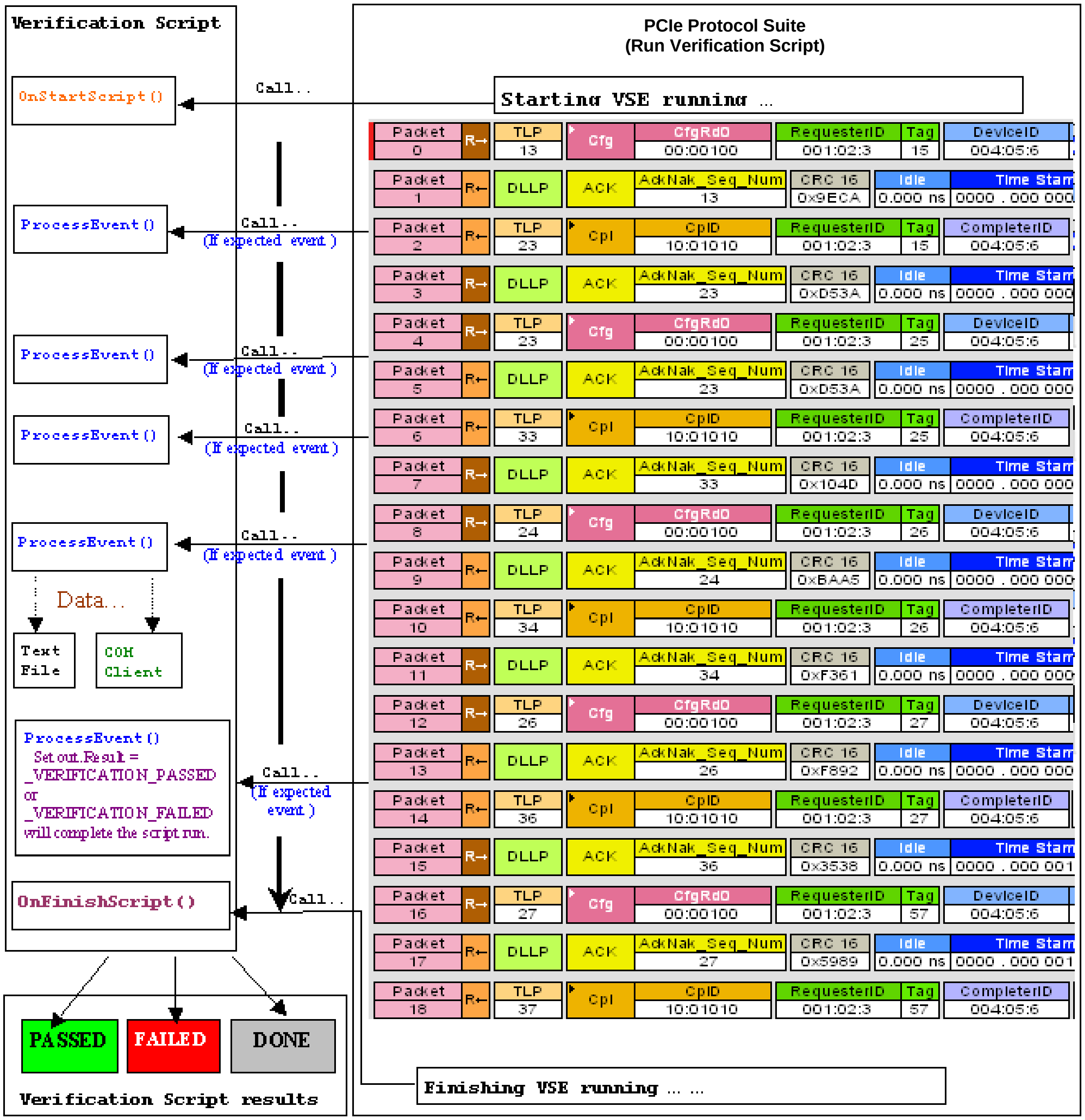
3. **ProcessEvent()** is the main verification routine for processing incoming trace events. This function must be present in all verification scripts. When the verification program consists of a few stages, the **ProcessEvent()** function processes the event sent to the script, verifies that information contained in the event is appropriate for the current stage, and decides if VSE should continue running the script or, if the whole result is clear on the current stage, tell VSE to complete execution of the script.
   The completion of the test before the entire trace has been evaluated is usually done by setting the output context variable in this manner:
   out.Result = _VERIFICATION_PASSED or **_VERIFICATION_FAILED**.

   (Please refer to the topic **Output context variables** later in this document for a full description of verification script output context variables)

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

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

The following figure illustrates the interaction between the PCIe Protocol Suite™ application and a running verification script:

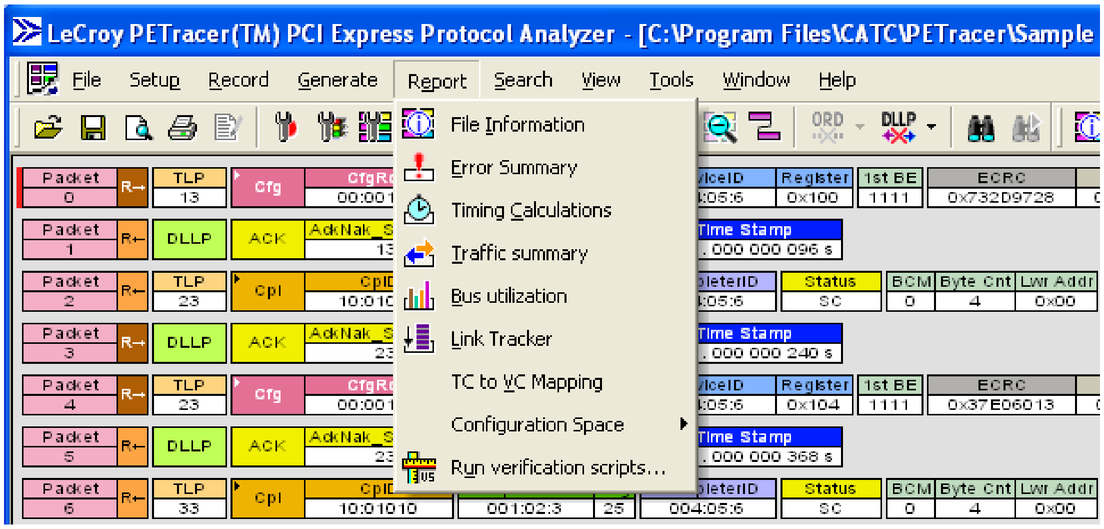


Packet 0	R→	TLP 13	Cfg	CfgRd0 00:00100	RequesterID 001:02:3	Tag 15	DeviceID 004:05:6
Packet 1	R←	DLLP	ACK	AckNak_Seq_Num 13	CRC 16 0x9ECA	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 2	R←	TLP 23	Cpl	CplD 10:01010	RequesterID 001:02:3	Tag 15	CompleterID 004:05:6
Packet 3	R→	DLLP	ACK	AckNak_Seq_Num 23	CRC 16 0xD53A	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 4	R→	TLP 23	Cfg	CfgRd0 00:00100	RequesterID 001:02:3	Tag 25	DeviceID 004:05:6
Packet 5	R←	DLLP	ACK	AckNak_Seq_Num 23	CRC 16 0xD53A	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 6	R←	TLP 33	Cpl	CplD 10:01010	RequesterID 001:02:3	Tag 25	CompleterID 004:05:6
Packet 7	R→	DLLP	ACK	AckNak_Seq_Num 33	CRC 16 0x104D	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 8	R→	TLP 24	Cfg	CfgRd0 00:00100	RequesterID 001:02:3	Tag 26	DeviceID 004:05:6
Packet 9	R←	DLLP	ACK	AckNak_Seq_Num 24	CRC 16 0xBAA5	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 10	R←	TLP 34	Cpl	CplD 10:01010	RequesterID 001:02:3	Tag 26	CompleterID 004:05:6
Packet 11	R→	DLLP	ACK	AckNak_Seq_Num 34	CRC 16 0xF361	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 12	R→	TLP 26	Cfg	CfgRd0 00:00100	RequesterID 001:02:3	Tag 27	DeviceID 004:05:6
Packet 13	R←	DLLP	ACK	AckNak_Seq_Num 26	CRC 16 0xF892	Idle 0.000 ns	Time Stam 0000 . 000 000
Packet 14	R←	TLP 36	Cpl	CplD 10:01010	RequesterID 001:02:3	Tag 27	CompleterID 004:05:6
Packet 15	R→	DLLP	ACK	AckNak_Seq_Num 36	CRC 16 0x3538	Idle 0.000 ns	Time Stam 0000 . 000 001
Packet 16	R→	TLP 27	Cfg	CfgRd0 00:00100	RequesterID 001:02:3	Tag 57	DeviceID 004:05:6
Packet 17	R←	DLLP	ACK	AckNak_Seq_Num 27	CRC 16 0x5989	Idle 0.000 ns	Time Stam 0000 . 000 001
Packet 18	R←	TLP 37	Cpl	CplD 10:01010	RequesterID 001:02:3	Tag 57	CompleterID 004:05:6

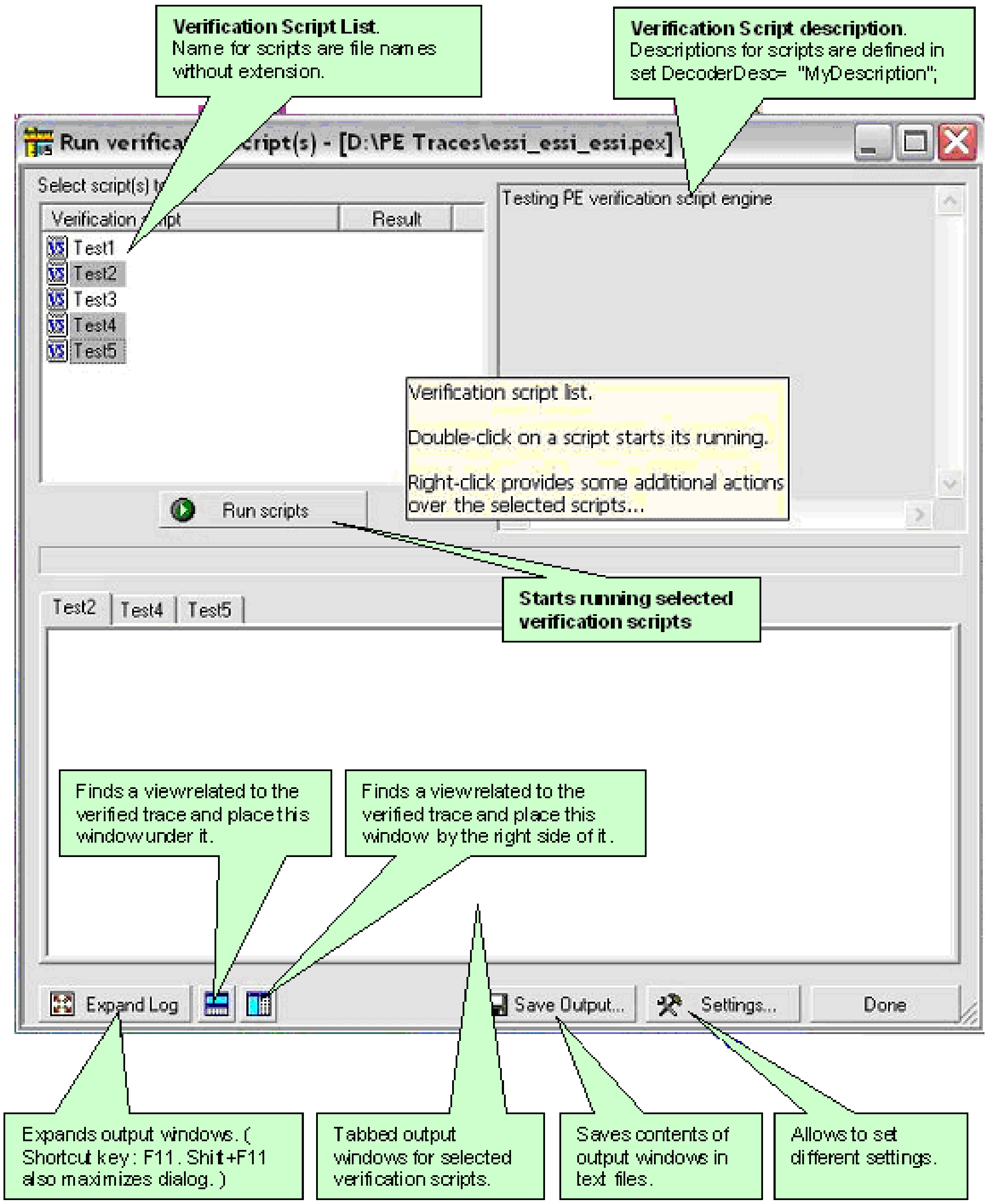
Note: The Verification script result "DONE" occurs when the script has been configured to extract and display some information about the trace, but not to display PASSED/FAILED results. To configure a script so that it only displays information – place a call somewhere in your script to the function **ScriptForDisplayOnly()in OnStartScript()**, for example.

4 Running verification scripts from the PCIe Protocol Suite

In order to run a verification script over a trace - you need to open the PCIe Protocol Suite™ main menu item **Report\Run verification scripts**… or push the icon on the main toolbar if it is not hidden.


LeCroy PETracer(TM) PCI Express Protocol Analyzer - [C:\Program Files\CATC\PETracer\Sample
File Setup Record Generate Report Search View Tools Window Help
File Information
Error Summary
Timing Calculations
Traffic summary
Bus utilization
Link Tracker
TC to VC Mapping
Configuration Space
Run verification scripts…

The special dialog opens displaying a list of verifications scripts. You can select one script to run, or several scripts from the list to run in parallel:



Verification Script List. Name for scripts are file names without extension.

Verification Script description. Descriptions for scripts are defined in set DecoderDesc= "MyDescription";

Verification script list.

Double-click on a script starts its running.

Right-click provides some additional actions over the selected scripts...

Starts running selected verification scripts

Finds a view related to the verified trace and place this window under it.

Finds a view related to the verified trace and place this window by the right side of it.

Expands output windows. (Shortcut key: F11. Shift+F11 also maximizes dialog.)

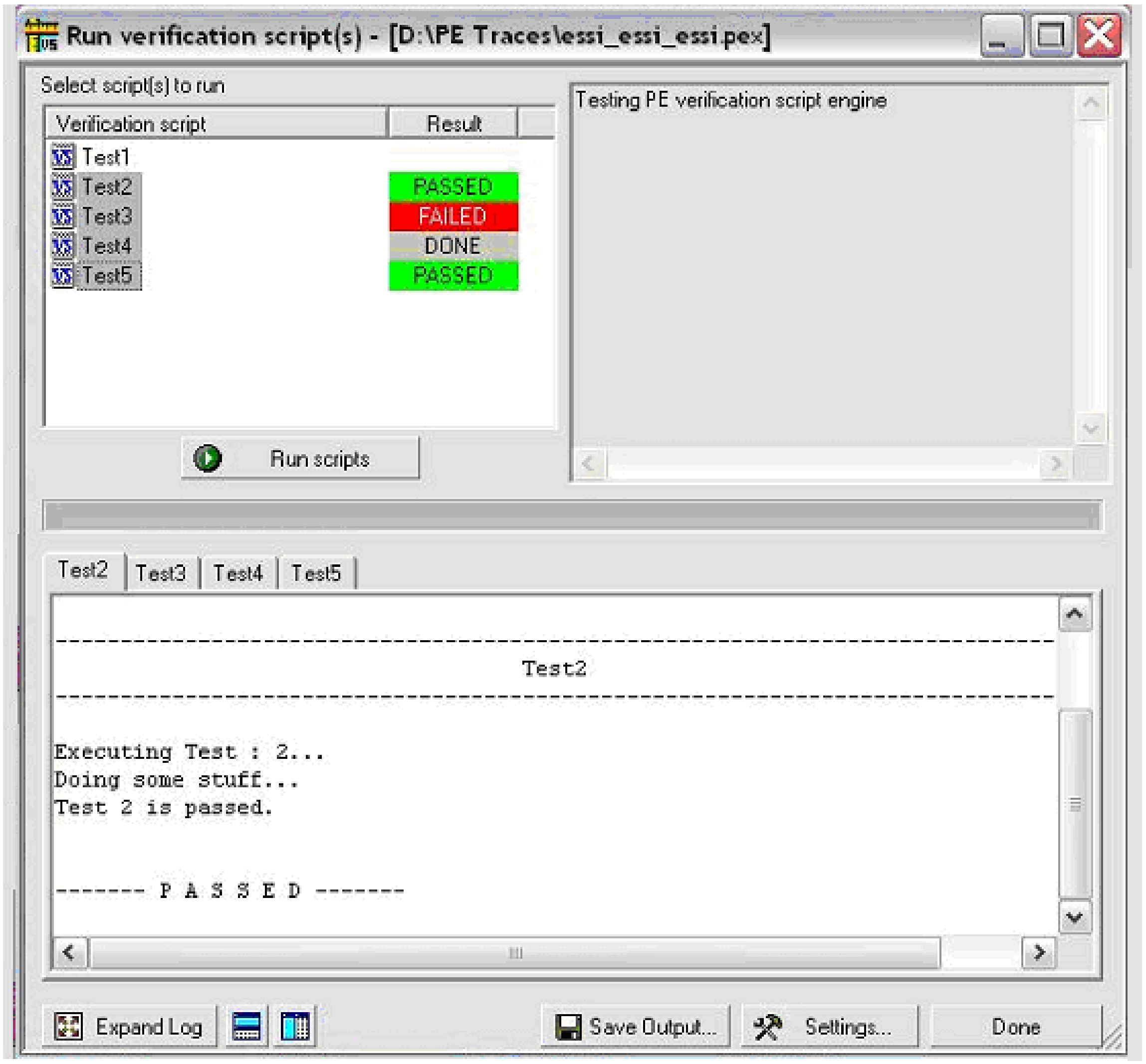
Tabbed output windows for selected verification scripts.

Saves contents of output windows in text files.

Allows to set different settings.

4.1 Running verification scripts

Push the button **Run scripts** after you selected the desired script(s) to run. VSE starts running the selected verification script(s), show script report information in the output windows, and present results of verifications in the script list:



Verification script	Result
Test1	
Test2	PASSED
Test3	FAILED
Test4	DONE
Test5	PASSED

Testing PE verification script engine

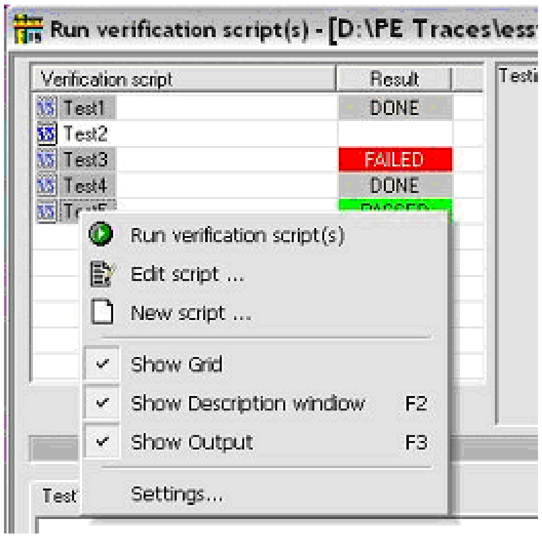
Test2 | Test3 | Test4 | Test5

```
------------------------------------------------------------------------------
                                   Test2
------------------------------------------------------------------------------

Executing Test : 2...
Doing some stuff...
Test 2 is passed.


------- P A S S E D -------
```

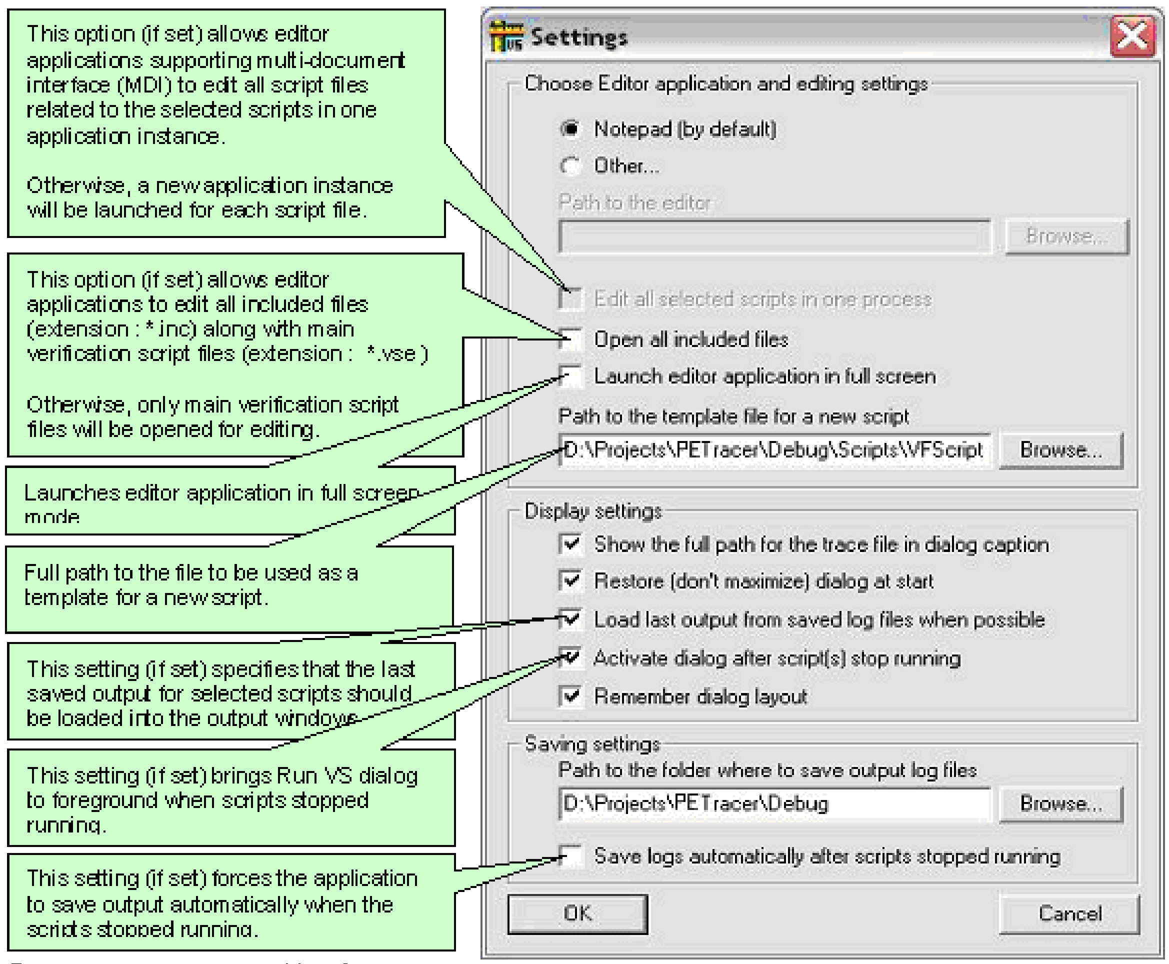
Right-click in script list opens a pop-up menu with options for performing additional operations on the selected scripts:



- **Run verification script(s)**: Starts running selected script(s).
- **Edit script**: Allows editing of the selected script(s) using whatever editor was specified in Editor settings.
- **New script**: Creates a new script file using the template specified in Editor settings.
- **Show Grid**: Shows/hides a grid in the verification script list**.**
- **Show Description window:** Shows/hides the script description window**. (Shortcut key : F2)**
- **Show Output:** Shows/hides the script output windows. **(Shortcut key : F3)**
- **Settings**: Opens a special Setting dialog which allows you to specify different settings for VSE.

4.2 VSE GUI Settings

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



This option (if set) allows editor applications supporting multi-document interface (MDI) to edit all script files related to the selected scripts in one application instance.

Otherwise, a new application instance will be launched for each script file.

This option (if set) allows editor applications to edit all included files (extension : *.inc) along with main verification script files (extension : *.vse)

Otherwise, only main verification script files will be opened for editing.

Launches editor application in full screen mode

Full path to the file to be used as a template for a new script.

This setting (if set) specifies that the last saved output for selected scripts should be loaded into the output windows

This setting (if set) brings Run VS dialog to foreground when scripts stopped running.

This setting (if set) forces the application to save output automatically when the scripts stopped running.

Settings

Choose Editor application and editing settings

- [✓ Notepad (by default)]
- Other...
- Path to the editor: (empty) Browse...
- [ ] Edit all selected scripts in one process
- [ ] Open all included files
- [ ] Launch editor application in full screen
- Path to the template file for a new script: D:\Projects\PETracer\Debug\Scripts\VFScript Browse...

Display settings

- [✓] Show the full path for the trace file in dialog caption
- [✓] Restore (don't maximize) dialog at start
- [✓] Load last output from saved log files when possible
- [✓] Activate dialog after script(s) stop running
- [✓] Remember dialog layout

Saving settings

- Path to the folder where to save output log files: D:\Projects\PETracer\Debug Browse...
- [ ] Save logs automatically after scripts stopped running

OK Cancel

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

5 Verification Script Engine Input Context Members

All verification scripts have input contexts – some special structures whose members are filled by the application and can be used inside of the scripts (for more details about input contexts – please refer to the *CATC Script Language(CSL) Manual*). The verification script input contexts have two sets of members:

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

5.1 Trace event-independent set of members

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

in.Level: Transaction level of the trace event (0 = packets, 1 = link transactions, 2 = split transactions, 3 = NVM transactions, 5 = AHCI transactions, 6 = ATA transactions, 9 = NVM commands)

in.Index: Index of the event in the trace file (frame number for frames, sequence number for sequences)

in.Time: Time of the event (type: list, having the format: 2 sec 125 ns -> [2 , 125]. (See 9.1 VSE Time Object for details)

in.Channel: Channel where the event occured. (may be **_CHANNEL_1** (1) or **_CHANNEL_2** (2) indicating which direction of the PE link the event occurred)

in.TraceEvent: Type of trace event (application predefined constants are used. See the list of possible events, below)

in.Notification: Type of notification (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 events or after calling some functions filling out some of the variables:

5.2.1 All packet/transaction-specific set of members

Members of this set are valid for any event.

in.Payload: Bit source of the frame/sequence payload (you can extract any necessary information using the **GetNBits(), NextNBits(),** or **PeekNBits()** functions. Refer to the ***CSL Manual*** for details about these functions.

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

in.LinkWidth: Link Width recorded for this packet. Possible values 1, 2, 4, 8, and 16 represent the number of lanes on the link. Only available at the Packet and Link Transaction levels.

in.Speed: Speed of this packet or link transaction: 0 = 2.5GT/s, 1 = 5.0 GT/s, or 2 = 8.0 GT/s. The following constants are defined for the possible values **_SPEED_GEN1, _SPEED_GEN2**, and **_SPEED_GEN3**.
Only available at the Packet and Link Transaction levels.

Error-related Variables (used for passing all the detected packet error types to the script)

in.HasErrors: Indicates the presence of any general error type in the current packet or critical packet-type-specific errors. It is a logical OR of **in.ErrorDisparity**, **in.ErrorSymbol, in.ErrorDelimiter, in.ErrorEndBad, in.ErrorAllignment, in.ErrorLength, in.ErrorWrongSymbol, in.BadLCRC** [TLP and DLLP Packet Types], **in.BadECRC** [TLP Packet Type], **in.MsgErrorG3LenCheck** [TLP Packet Type], and **in.G3ErrorFraming** [DLLP Packet Type]. If this variable is 1, one or more of the errors indicated are present. If it is 0 (zero), the errors indicated are not present.

in.ErrorDisparity: If set to a non-zero value, indicates presence of Running Disparity error(s) in this packet.

in.ErrorSymbol: If set to a non-zero value, indicates presence of Symbol (10-bit Code) error(s) in this packet.

in.ErrorWrongSymbol: If set to a non-zero value, indicates a K symbol was received where a D symbol was expected, or vice versa.

in.ErrorDelimiter: If set to a non-zero value, indicates presence of Delimiter error(s) in this packet.

in.ErrorEndBad: If set to a non-zero value, indicates presence of an EDB symbol in this packet.

in.ErrorAlignment: If set to a non-zero value, indicates presence of Alignment error(s) in this packet.

in.ErrorLength: If set to a non-zero value, indicates presence of Bad Length error(s) in this TLP packet.

in.HasIdleErrors: Indicates presence of Idle errors in the current packet. If set, one of the following is set, indicating the presence of error(s) between this packet and the previous packet on this direction of the link:

in.IdleErrorDisparity: If set to a non-zero value, indicates presence of Running Disparity error(s).

in.IdleErrorSymbol: If set to a non-zero value, indicates presence of Symbol (10-bit Code) error(s).

in.IdleErrorSkip: If set to a non-zero value, indicates presence of Skip error(s).

in.IdleErrorData: If set to a non-zero value, indicates presence of Logical Idle data pattern error(s).

Note: For CRC error variables, see the specific packet type variable sets below.

5.2.2 DLLP-specific set of members

Valid for data link layer packets only. Undefined for other events.

in.DLLPType: Contains the numeric encoding of the DLLP type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

```
DLLP_TYPE_ACK                    = 0x0;
DLLP_TYPE_NAK                    = 0x1;
DLLP_TYPE_INIT_FC1_P             = 0x4;
DLLP_TYPE_INIT_FC1_NP            = 0x5;
DLLP_TYPE_INIT_FC1_CPL           = 0x6;
DLLP_TYPE_INIT_FC2_P             = 0xC;
DLLP_TYPE_INIT_FC2_NP            = 0xD;
DLLP_TYPE_INIT_FC2_CPL           = 0xE;
DLLP_TYPE_UPDATE_FC_P            = 0x8;
DLLP_TYPE_UPDATE_FC_NP           = 0x9;
DLLP_TYPE_UPDATE_FC_CPL          = 0xA;
DLLP_TYPE_VENDOR                 = 0x3;

DLLP_TYPE_PM_ENTER_L1            = 0x10;
DLLP_TYPE_PM_ENTER_L23           = 0x11;
DLLP_TYPE_PM_ACT_STATE_REQUEST_L1 = 0x13;
DLLP_TYPE_PM_REQUEST_ACK         = 0x14;

DLLP_TYPE_INVALID                = 0x7;
```

in.AckNak_SeqNum: Field value (valid only for Ack and Nak DLLPs), indicating which TLPs are affected by the acknowledgement

in.VC_ID: Virtual Channel ID (valid only for InitFC and UpdateFC DLLPs)

in.HdrFC: Credit value for headers of the type indicated by the DLLP type (valid only for InitFC and UpdateFC DLLPs)

in.DataFC: Credit value for payload data of the type indicated by the DLLP type (valid only for InitFC and UpdateFC DLLPs)

in.VendorSpecific: 3-byte vendor-defined value in a Vendor-specific DLLP

DLLP Error related Variables

in.InvalidEncoding: If set to a non-zero value, indicates an invalid DLLP encoding.

in.RsvdField: If set to a non-zero value, indicates a reserved field is non-zero or in use.

in.G3ErrorFraming: If set to a non-zero value, indicates Symbol 1 is incorrect, but Symbol 0 is correct. The value of the incorrect Symbol 1 is stored in **in.G3ErrorSym1Val**.
Note: This is a PCIE Gen 3 DLLP error.

in.G3ErrorSym1Val: The value of an incorrect Symbol 1 if **in.G3ErrorFraming** is set to a non-zero value.

in.FCError: If set to a non-zero value, indicates a Flow Control initialization protocol violation.

in.BadCRC: Set to 1 if the DLLP has bad 16-bit CRC and to 0 otherwise.

Un-decoded Frame

in.Frame: Contains the complete DLLP frame, i.e. from SDP till END.

5.2.3 TLP-specific set of members

Valid for TLPs only, undefined for other events.

All TLPs

in.TLPType: Contains the numeric encoding of the TLP type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

```
TLP_TYPE_ID_INVALID = 0;
TLP_TYPE_ID_MRD32   = 1;
TLP_TYPE_ID_MRDLK32 = 2;
TLP_TYPE_ID_MWR32   = 3;
TLP_TYPE_ID_MRD64   = 4;
TLP_TYPE_ID_MRDLK64 = 5;
TLP_TYPE_ID_MWR64   = 6;
TLP_TYPE_ID_IORD    = 7;
TLP_TYPE_ID_IOWR    = 8;
TLP_TYPE_ID_CFGRD_0 = 9;
TLP_TYPE_ID_CFGWR_0 = 10;
TLP_TYPE_ID_CFGRD_1 = 11;
TLP_TYPE_ID_CFGWR_1 = 12;
TLP_TYPE_ID_MSG     = 13;
TLP_TYPE_ID_MSGD    = 14;
TLP_TYPE_ID_MSGAS   = 15;
TLP_TYPE_ID_MSGASD  = 16;
TLP_TYPE_ID_CPL     = 17;
TLP_TYPE_ID_CPLD    = 18;
TLP_TYPE_ID_CPLLK   = 19;
TLP_TYPE_ID_CPLDLK  = 20;
```

Note: For a comprehensive and most up to date list of constants and codes please review file \Users\Public\Documents\LeCroy\PCIe Protocol Suite\Scripts\VFScripts\VS_constants.inc

TLP Error-related Variables

in.InvalidEncoding: If set to a non-zero value, indicates an invalid TLP encoding.

in.ErrorRsvdFld: If set to a non-zero value, indicates a reserved field is non-zero or in use.

in.ErrorPayload: If set to a non-zero value, indicates the TLP Payload does not match the Length field, so that the TD field value does not correspond with the observed size.

in.ErrorLengthField: If set to a non-zero value, indicates the Length field is invalid.

in.ErrorTCField: If set to a non-zero value, indicates the TC field is invalid.

in.ErrorAttrField: If set to a non-zero value, indicates the Attr field is invalid.

in.ErrorByteEnables: If set to a non-zero value, indicates the TLP violates the Byte Enable rules.

in.MemErrorAddrLength: If set to a non-zero value, indicates the Address/Length combination causes a Memory Space access to cross a 4-KB boundary.

in.MemErrorWrongType: If set to a non-zero value, indicates the wrong bit format is being used. For example, for addresses below 4 GB, Requesters must use 32 bit format.

in.CfgErrorRegister: If set to a non-zero value, indicates an invalid register field for Cfg. Must be DWORD aligned.

in.MsgErrorRouting: If set to a non-zero value, indicates invalid Msg or MsgD routing.

in.MsgErrorG3LenCheck: If set to a non-zero value, indicates a CRC-4 and/or Parity check failed on a Gen3 TLP length field (in framing, not the header). **Note**: This is a PCIE Gen 3 error.

in.BadLCRC: Set to 1 if the TLP has bad LCRC, to 0 otherwise

in.BadECRC: Set to 1 if the TLP has bad ECRC (when it should be present), to 0 otherwise

Field values for all TLP types:

in.Type: Type of TLP field value

in.Fmt: Format of TLP field value

in.PSN: Packet Sequence Number for this TLP as set by the Data Link Layer

in. RequesterId: Requester ID value (Bus, Device and Function Number fields combined)

in.Tag: Tag field value

in.TC: Traffic Class field value

in.Snoop: Snoop attribute bit value

in.Ordering: Ordering attribute bit value

in.IDBasedOrdering: ID Based Ordering attribute bit value

in.Attributes: Attributes field value [all three bits (Snoop, Ordering, and IDBasedOrdering)]

in.TH: TLP Processing Hints bit value

in.TD: TLP Digest bit value

in.EP: Poisoned TLP bit value

in.AT: Address Type field value

in.Length: Length field value

in.LCRC: LCRC value as set by the Data Link Layer

in.ECRC: ECRC value (optional)

Field values dependant upon TLP type:

in.FirstDwBe: Byte Enable bits for the first DW of the payload (all TLPs except Completions and Messages)

in.LastDwBe: Byte Enable bits for the last DW of the payload (all TLPs except Completions and Messages)

in.Address: 32-bit Address value for IO, Configuration, and Mem-32 requests

in.AddressLo: Low 32 bits of the Address for Mem-64 requests and Messages routed by address

in.AddressHi: High 32 bits of the Address for Mem-64 requests and Messages routed by address

in.DeviceId: Requester ID value (Bus, Device and Function Number fields combined) for Configuration requests and Messages routed by ID

in.Register: Register address (Register Number and Extended Register Number combined) for Configuration requests

For Completion TLPs only:

in.CompleterId: Completer ID value (Bus, Device and Function Number fields combined)

in.ComplStatus: Completion Status field value

in.BCM: Byte Count Modified bit value

in.ByteCount: Remaining Byte Count field value

in.LowerAddr: Lower Address for starting byte of completion field value

For Message TLPs only:

in.MessageCode: Message Code field value

in.MessageRoute: Message Routing field value (from the TLP Type field)

Note: For a comprehensive and most up to date list of constants and codes please review file \Users\Public\Documents\LeCroy\PCIe Protocol Suite\Scripts\VFScripts\VS_constants.inc

For Configuration Write Requests and Read Completions:

in.RegisterData: 32-bit value written to or read from a configuration register (for convenience of processing the configuration requests, as it also can be obtained from the Payload)

Un-decoded Frame

in.Frame: Contains the complete TLP frame, i.e. STP till END.

5.2.4 Ordered Set specific set of members

in.OrderedSetType: Contains the numeric encoding of the Ordered Set type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

```
ORDSET_TYPE_TS1       = 0x02;
ORDSET_TYPE_TS2       = 0x03;
ORDSET_TYPE_FTS       = 0x04;
ORDSET_TYPE_EIOS      = 0x05;
ORDSET_TYPE_SKIP      = 0x06;
ORDSET_TYPE_PATN      = 0x07;
ORDSET_TYPE_EIEOS     = 0x08;
ORDSET_TYPE_SDS       = 0x0C;
```

Note: For a comprehensive and most up to date list of constants and codes please review file \Users\Public\Documents\LeCroy\PCIe Protocol Suite\Scripts\VFScripts\VS_constants.inc

For Training Sequences (TS1 and TS2), the following variables of the list type exist in the input context (the lists are arrays of integers with dimensions equal to the Link Width for the Training Sequence packet).

in.TS_LinkNumberList: Contains the Link Number parameter values for all lanes

in.TS_LaneNumberList: Contains the Lane Number parameter values for all lanes

in.TS_N_FTSList: Contains the N_FTS parameter values for all lanes

in.TS_TrainingControlList: Contains the Training Control bitmap parameter values for all lanes

in.TS_RawSymbolsList: Contains the list of all symbols for all lanes

in.TS_DataRateList: Contains the Data Rate parameter values for all lanes

The following parameters are only valid for Gen 3.

in.TS_PreCursorList: Contains the Pre-Cursor parameter values for all lanes

in.TS_CursorList: Contains the Cursor parameter values for all lanes

in.TS_PostCursorList: Contains the Post-Cursor parameter values for all lanes

Note: For Link Number and Lane Number values the special value of 0x1FF is used to indicate the PAD symbol. Please refer to the **examp_ordered_sets.pevs** sample script for an example of how to process ordered Sets and Training Sequences, in particular.

5.2.5 Link Condition specific set of members

in.LinkConditionType: Contains the numeric encoding of the Link Condition type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

LINK_CONDITION_UNKNOWN	=0;	- Link Condition unknown.
LINK_CONDITION_LINK_UP	=1;	- “Link Up” Link Condition event
LINK_CONDITION_LINK_DOWN	=2;	- “Link Down” Link Condition event
LINK_CONDITION_SKEW	=3;	- “Deskewing” Link Condition event
LINK_CONDITION_LINK_WAKE_UP	=4;	- "Link Wake Up" Link Condition event
LINK_CONDITION_LINK_WAKE_DOWN	=5;	- "Link Wake Down" Link Condition event

Note: For a comprehensive and most up to date list of constants and codes please review file \Users\Public\Documents\LeCroy\PCIe Protocol Suite\Scripts\VFScripts\VS_constants.inc

5.2.6 Link transaction-specific set of members

Valid for Link transactions only, undefined for other events.

All the TLP-specific values are present in the input context for Link transactions, depending upon the type of TLP for this Link transaction. In addition to that, the following value exists:

in.TransactionStatus: Status for this Link transaction. Can be one of three values: Implicitly Acknowledged, Explicitly Acknowledged, or Incomplete (Link Layer error). See file **VS_constants.inc** for encodings.

Metric values

The following values are defined in input context for Link Transactions that are related to Unit Metrics. To learn more about Unit Metrics, please refer to PCIe Protocol Suite™ Help.

in.Metric_NumOfPackets: Metric presenting the total number of packets that compose this Link Transaction, an integer value

in.Metric_ResponseTime: Metric presenting time it took to transmit this Link Transaction on the PE link, from the beginning of the first packet in the transaction to the end of the last packet in the transaction, a VSE time object value (see 9.1 VSE Time Object for details)

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this Link Transaction transferred, an integer value

Notes: For the incomplete Link Transactions only, the NumOfPackets metric is valid. In case of an incomplete Link Transaction, the ResponseTime metric value is set to `null`.

5.2.7 Split transaction-specific set of members

Valid for Split transactions only. Undefined for other events.

All the TLP-specific values **for the request TLP of the split transaction** are present in the input context for Link transactions, depending upon the type of TLP for this Link transaction. Also the common PayloadLength and Payload values reflect the total combined payload for the Split transaction. In addition to that, the following values exist:

in.CompletionStatus: Completion Status for this Split transaction. From the last completion of the response.

Metric values

The following values are defined in input context for Split Transactions that are related to Unit Metrics. To learn more about Unit Metrics please refer to PCIe Protocol Suite Help.

in.Metric_NumOfPackets: Metric presenting the total number of packets that compose this Link Transaction, an integer value

in.Metric_ResponseTime: Metric presenting time it took to transmit this Split Transaction on the PE link, from the beginning of the first packet in the transaction to the end of the last packet in the transaction, a VSE time object value (see 9.1 VSE Time Object for details)

in.Metric_LatencyTime: Metric presenting time measured from the end of the request transaction to the first completion transmitted in response to the request within this Split Transaction, a VSE time object value (see 9.1 VSE Time Object for details)

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this Split Transaction transferred, an integer value.

Notes: For the incomplete Link Transactions only, the NumOfPackets metric is valid. In case of an incomplete Link Transaction the ResponseTime metric value is set to `null`.

5.2.8 NVM transaction-specific set of members

Valid for NVM transactions only. Undefined for other events.

in.nvmeType: Returns NVMe register type. The value of ‘in.nvmeType’ depends on transaction event type and can be compared against the predefined values. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

For _NVME_CONTROLLER_REG transaction event:

Constant	**Value**
_NVME_CAP	0
_NVME_VS	1
_NVME_INTMS	2
_NVME_INTMC	3
_NVME_CC	4
_NVME_RESERVED1	5
_NVME_CSTS	6
_NVME_NSSR	7
_NVME_AQA	8
_NVME_ASQ	9
_NVME_ACQ	10
_NVME_RESERVED2	11
_NVME_RESERVED_CMD_SET_SPECIFIC	12

For _NVME_DOORBELL_REG transaction event:

Constant	**Value**
_NVME_ADMIN_SQTDBL	14
_NVME_ADMIN_CQHDBL	15
_NVME_SQYTDBL	16
_NVME_CQYHDBL	17

For _NVME_ADMIN_SUBMISSION_CMD transaction event:

Constant	**Value**
_NVME_ADMIN_SUBMISSION_Q_ENTRY	18

For _NVME_COMPLETION_CMD transaction event:

Constant	**Value**
_NVME_ADMIN_COMPLETION_Q_ENTRY	19

For _NVME_NVM_SUBMISSION_CMD transaction event:

Constant	**Value**
_NVME_IO_SUBMISSION_Q_ENTRY	20

For _NVME_COMPLETION_CMD transaction event:

Constant	**Value**

_NVME_IO_COMPLETION_Q_ENTRY	21

For _NVME_PRP transaction event:

Constant	Value
_NVME _CMD_PRP	80
_NVME _CMD_PRP_LIST	81

For _NVME_SGL transaction event:

Constant	Value
_NVME_SGL_DESCRIPTOR	83
_NVME _MSGLP	84

For _NVME_TRANSFERED_DATA transaction event:

Constant	Value
_NVME_DATA	86

For _NVME_IDX_DAT_REG transaction event:

Constant	Value
_NVME_IDX	87
_NVME_DAT	88

in. nvmeQID: Defined for _NVME_DOORBELL_REG transaction event. Returns zero for Admin doorbells, or queue ID otherwise.

in. nvmeIndex: Defined for _NVME_DOORBELL_REG transaction event. Returns SQT for submission doorbells, and CQH – for completions.

in.nvmeTraHasError: If set to a non-zero value, indicates NVME transaction has errors.

In.nvmeErrorId: Contains the numeric encoding of the NVME error type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Error type	Value	Error name	Error description
_NVME_ERROR_ACCESS_DIR_VIOLATION	1	Read-only registry write	Access direction violation
_NVME_ERROR_RESERVED_NOT_NULL	2	Reserved field is not zero	Reserved field is not zero
_NVME_ERROR_INVALID_FIELD_VALUE	4	Field value is not from specified set	Field value is not listed as a valid value
_NVME_ERROR_INCOMPLETE_TRA	16	Incomplete transaction	Incomplete transaction: size doesn't match expected
_NVME_ERROR_INCOMPLETE_SUB_TRA	32	Incomplete sub-transaction	Incomplete sub-transaction
_NVME_ERROR_ERROR_IN_SUB_TRA	64	Error in sub-transaction	Error in sub-transaction
_NVME_ERROR_LOGICAL_ERROR	128	Logical error	Logical error
_NVME_ERROR_NO_ERROR	0	No Error	Correct transaction

in.nvmeErrorIdAsString: Contains an NVME error name

5.2.8.1 NVM transaction members specific to _NVME_ADMIN_SUBMISSION_CMD and to NVME_NVM_SUBMISSION_CMD events:

in.nvmeCID: Returns Command Id.

in.nvmePSDT: Returns whether PRPs or SGLs are used for any data transfer associated with the command. If cleared to ‘0’, the command uses PRPs.

in.nvmeFUSE: In a fused operation, returns whether complex command is created by “fusing” together two simpler commands.

in. nvmeOpcode: Returns the NVMe command code. The value of ‘in.nvmeOpcode’ depends on transaction event type and can be compared against the predefined values. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

For _NVME_ADMIN_SUBMISSION_CMD transaction event:

Constant	**Value**
_NVME_ASC_DELETE_IO_SQ	00h
_NVME_ASC_CREATE_IO_SQ	01h
_NVME_ASC_GET_LOG_PAGE	02h
_NVME_ASC_DELETE_IO_CQ	04h
_NVME_ASC_CREATE_IO_CQ	05h
_NVME_ASC_IDENTIFY	06h
_NVME_ASC_ABORT	08h
_NVME_ASC_SET_FEATURES	09h
_NVME_ASC_GET_FEATURES	0Ah
_NVME_ASC_ASYNC_EVENT_REQ	0Ch
_NVME_ASC_FIRMWARE_ACTIVATE	10h
_NVME_ASC_FIRMWARE_IMG_DWNLD	11h
_NVME_ASC_FORMAT_NVM	80h
_NVME_ASC_SECURITY_SEND	81h
_NVME_ASC_SECURITY_RECEIVE	82h

For _NVME_NVM_SUBMISSION_CMD transaction event:

Constant	**Value**
_NVME_NSC_FLUSH	00h
_NVME_NSC_WRITE	01h
_NVME_NSC_READ	02h
_NVME_NSC_WRITE_UNCORRECTABLE	04h
_NVME_NSC_COMPARE	05h
_NVME_NCS_WRITE_ZEROES	08h
_NVME_NSC_DATASET_MGMT	09h
_NVME_NSC_RESERVATION_REGISTER	0Dh
_NVME_NSC_RESERVATION_REPORT	0Eh
_NVME_NSC_RESERVATION_ACQUIRE	11h
_NVME_NSC_RESERVATION_RELEASE	15h

in.nvme NSID: This field specifies the namespace ID that this command applies to.

in.nvme MPTR: Returns the address of a contiguous physical buffer of metadata or the address of an SGL segment containing exactly one SGL Descriptor which describes the metadata to transfer

in.nvme DPTR: This field specifies the data used in the command.

in.nvme CDW10: Returns command-specific Dword #10.

in.nvme CDW11: Returns command-specific Dword #11.

in.nvme CDW12: Returns command-specific Dword #12.

in.nvme CDW13: Returns command-specific Dword #13.

in.nvme CDW14: Returns command-specific Dword #14.

in.nvme CDW15: Returns command-specific Dword #15.

5.2.8.2 NVM transaction members specific to _NVME_NVM_COMPLETION_CMD event

in.nvmeCDW0: Returns command-specific Dword #0.

in.nvmeCDW1: Returns command-specific Dword #1.

in.nvmeSQID: Returns the Submission Queue to which the associated command was issued to.

in.nvmeSQHD: Returns the current Submission Queue Head pointer for the Submission Queue indicated in the SQ Identifier field.

in.nvmeSF: Returns the status for the command that is being completed.

in.nvmeP: Returns Phase Tag (P) (identifies whether a Completion Queue entry is new).

in.nvmeCID: Returns Command Id.

in.nvmeQID: Returns queue Id.

5.2.8.3 NVM transaction members specific to _NVME_PRP event

in.nvmeCID: Returns Command Id.

in.nvmeQID: Returns queue Id.

5.2.8.4 NVM transaction members specific to _NVME_SGL event

in.nvmeCID: Returns Command Id.

in.nvmeQID: Returns queue Id.

in.nvmeAddress: Returns address of the next SGL segment (64-bit value).

in.nvmeLength: Length of SGL segment.

in.nvmeSGLId: SGL identifier.

5.2.8.5 NVM transaction members specific to _NVME_TRANSFERED_DATA event

in.nvmeCID: Returns Command Id.

in.nvmeQID: Returns queue Id.

in.nvmeDataAddress: Returns SGL segment or PRP entry address.

in.nvmeDataLength: Length of SGL segment or PRP entry.

5.2.8.6 Metric values

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this NVM Transaction transferred, an integer value.

in.Metric_NumOfLinkAndSplitTras: Metric presenting the total number of Link and Split Transactions that compose this NVM Transaction, an integer value.

5.2.9 NVM command-specific set of members

Valid for NVM commands only. Undefined for other events.

All the NVM command-specific values are present in the input context for NVM commands. Also the common PayloadLength and Payload values reflect the total combined payload for the NVM command. In addition to that, the following values exist:

in.nvmcDeviceId: Returns command device id.

in.nvmcCommandOpCode: Returns command opcode. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Admin commands		
Command	**Value**	**Command name**
_NVMC_OPCODE_FLUSH	0x00	Flush
_NVMC_OPCODE_WRITE	0x01	Write
_NVMC_OPCODE_READ	0x02	Read
_NVMC_OPCODE_WRITE_UNCORRECTABLE	0x04	Write Uncorrectable
_NVMC_OPCODE_COMPARE	0x05	Compare
_NVMC_OPCODE_WRITE_ZEROES	0x08	Write Zeros
_NVMC_OPCODE_DATASET_MGMT	0x09	Dataset Management
_NVMC_OPCODE_RESERVATION_REGISTER	0x0D	Reservation Register
_NVMC_OPCODE_RESERVATION_REPORT	0x0E	Reservation Report
_NVMC_OPCODE_RESERVATION_ACQUIRE	0x11	Reservation Acquire
_NVMC_OPCODE_RESERVATION_RELEASE	0x15	Reservation Release
_NVMC_OPCODE_DATASET_MGMT_VENDOR_SPECIFIC_FIRST	0x80	Vendor Specific
_NVMC_OPCODE_DATASET_MGMT_VENDOR_SPECIFIC_LAST	0xFF	Vendor Specific
I / O commands		
_NVMC_OPCODE_DELETE_IO_SQ	0x00	Delete I/O Submission Queue
_NVMC_OPCODE_CREATE_IO_SQ	0x01	Create I/O Submission Queue
_NVMC_OPCODE_GET_LOG_PAGE	0x02	Get Log Page
_NVMC_OPCODE_DELETE_IO_CQ	0x04	Delete I/O Completion Queue
_NVMC_OPCODE_CREATE_IO_CQ	0x05	Create I/O Completion Queue
_NVMC_OPCODE_IDENTIFY	0x06	Identify
_NVMC_OPCODE_ABORT	0x08	Abort
_NVMC_OPCODE_SET_FEATURE	0x09	Set Feature
_NVMC_OPCODE_GET_FEATURE	0x0A	Get Feature
_NVMC_OPCODE_ASYNC_EVENT_REQUEST	0x0C	Asynchronous Event Request
_NVMC_OPCODE_FIRWARE_ACTIVATE	0x10	Firmware Activate
_NVMC_OPCODE_FIRMWARE_IMG_DOWNLOAD	0x11	Firmware Image Download
_NVMC_OPCODE_FORMAT_NVM	0x80	Format NVM
_NVMC_OPCODE_SECURITY_SEND	0x81	Security Send
_NVMC_OPCODE_SECURITY_RECEIVE	0x82	Security Receive
_NVMC_OPCODE_OTHER_IO_COMMAND_SET_SPECIFIC	0x83	Other IO command set specific
_NVMC_OPCODE_VENDOR_SPECIFIC_FIRST	0xC0	Vendor Specific
_NVMC_OPCODE_VENDOR_SPECIFIC_LAST	0xFF	Vendor Specific

in.nvmcSubmissionQueueID: Returns command submission queue id.

in.nvmcCompletionQueueID: Returns command completion queue id.

in.nvmcCommandID: Returns command id.

in.nvmcIsSecurityCommand: If set to a non-zero value, indicates NVM command is security.

in.nvmcIsDeviceToHostCommand: If set to a non-zero value, indicates NVM command transfers data from device to host.

in.nvmcIsSuccessful: If set to a non-zero value, indicates NVM command is successful.

in.nvmcStatus: Returns command status numeric encoding.

in.nvmcStatusType: Returns command status type numeric encoding.

in.nvmcIsIncomplete: If set to a non-zero value, indicates NVM command is incomplete.

in.nvmcHasInput: NVM command contains submission queue entry.

in.nvmcHasOutput: NVM command contains completion queue entry.

in.nvmcIsAdminCommand: If set to a non-zero value, indicates NVM command is admin.

in.nvmcNamespaceId: Returns namespace Id.

in.nvmcErrorId: Returns NVM error with the smallest numeric encoding. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Error type	Value	Error name	Error description
_NVMC_ERROR_NO_ERROR	0	No error	No errors
_NVMC_ERROR_INCOMPLETE_SUB_TRA	1	Incomplete Sub-Transaction	Incomplete Sub Transaction
_NVMC_ERROR_SUB_TRA_HAS_ERROR	2	Error in Sub-Transaction	Error in Sub Transaction
_NVMC_ERROR_INCOMPLETE_TRA	3	Incomplete Transaction	Incomplete command
_NVMC_ERROR_LOGICAL_ERROR	4	Logical Error	Logical error

in.nvmcErrorIdAsString: Contains an NVM error name.

in.nvmcTraHasError: If set to a non-zero value, indicates NVM command has errors.

in.nvmcNumBytesTransferred: Returns amount of transferred data by command in bytes.

in.nvmcNumBytesRequested: Returns number bytes requested by command.

in.nvmcUtilizesPRP: If set to a non-zero value, indicates that command uses PRP instead of SGL.

The following table shows the list of NVM commands and their fields defined in the input context. The fields can be accessed by using "_". E.g. **in. CreateIOCQ_PC** contains the numeric encoding of the PC field of Create I/O Completion Queue command. Some commands have repeating blocks(Power State Descriptors in Identify command), these blocks can be accesed the following way: <Command>_<Block_name><index>_<fileds>, i.e. **in.Identify_PSD3_MP**.

Note: If length of returned value is bigger than 1 dword, please, specify dword by using “_DW” and dword index, otherwise string in a hex format will be returned. For example **in. GetLogPage_POWER_CYCLES_DW0** contains the numeric encoding of the 1st dword of Power Cycles field of GetLogPage: SMART / Health Information Log command.

Submission Queue Entry Data

Command	Parameter	Fields	starting DWORD	Offset in bits	Meaning
Abort	Abort	SQID	10	15:0	Submission Queue Identifier
		CID	10	31:16	Command Identifier
Create I/O	CreateIOCQ	PRP1	6	63:0	PRP Entry 1

Completion Queue		QID	10	15:0	Queue Identifier
		QSIZE	10	31:16	Queue Size
		PC	11	0	Physically Contiguous
		IEN	11	1	Interrupts Enabled
		RSVD	11	15:2	Reserved
		IV	11	31:16	Interrupt Vector
Create I/O Submission Queue	CreateIOSQ	PRP1	6	63:0	PRP Entry 1
		QID	10	15:0	Queue Identifier
		QSIZE	10	31:16	Queue Size
		PC	11	0	Physically Contiguous
		QPRIO	11	2:1	Queue Priority
		RSVD	11	15:3	Reserved
		CQID	11	31:16	Completion Queue Identifier
Delete I/O Completion Queue	DeleteIOCQ	QID	10	15:0	Queue Identifier
		RSVD	10	31:16	Reserved
Delete I/O Submission Queue	DeleteIOSQ	QID	10	15:0	Queue Identifier
		QID	10	31:16	Reserved
Firmware Activate	FirmwareActivate	FS	10	2:0	Firmware Slot
		AA	10	4:3	Activate Action
		RSVD	10	31:05	Reserved
Firmware Image Download	FirmwareImageDownload	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		NUMD	10	31:0	Number of Dwords
		OFST	11	31:0	Offset
Get Features	GetFeatures	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		FID	10	7:0	Feature Identifier
		SEL	10	10:8	Select
		RSVD	10	31:11	Reserved
Get Log Page	GetLogPage	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		LID	10	7:0	Log Page Identifier
		RSVD	10	15:8	Reserved
		NUMD	10	27:16	Number of Dwords
		RSVD1	10	31:28	Reserved
Identify	Identify	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		CNS	10	1:0	Controller or Namespace Structure
		RSVD	10	31:2	Reserved
Set Features	SetFeatures	PRP1	6	63:0	PRP Entry 1

		PRP2	8	63:0	PRP Entry 2
		FID	10	7:0	Feature Identifier
		RSVD	10	30:8	Reserved
		SV	10	31	Save
Format NVM	FormatNVM	LBAF	10	3:0	LBA Format
		MS	10	4	Metadata Settings
		PI	10	7:5	Protection Information
		PIL	10	8	Protection Information Location
		SES	10	11:9	Secure Erase Settings
		RSVD	10	31:12	Reserved
Security Receive	SecurityReceive	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		RSVD	10	7:0	Reserved
		SPSP	10	23:8	SP Specific
		SECP	10	31:24	Security Protocol
		AL	11	31:0	Allocation Length
Security Send	SecuritySend	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		RSVD	10	7:0	Reserved
		SPSP	10	23:8	SP Specific
		SECP	10	31:24	Security Protocol
		TL	11	31:0	Transfer Length
Read	Read	MPTR	4	63:0	Metadata Pointer
		PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		MSGLP	4	63:0	Metadata SGL Segment Pointer
		SGL1	6	63:0	SGL Entry 1
		SLBA	10	63:0	Starting LBA
		NLB	12	15:0	Number of Logical Blocks
		RSVD	12	25:16	Reserved
		PRINFO	12	29:26	Protection Information Field
		FUA	12	30	Force Unit Access
		LR	12	31	Limited Retry
		ACCF	13	3:0	Access Frequency
		ACCL	13	5:4	Access Latency
		SEQR	13	6	Sequential Request
		INCOM	13	7	Incompressible
		RSVD1	13	10:8	Reserved
		EILBRT	14	31:0	Expected Initial Logical Block Reference Tag
		ELBAT	15	15:0	Expected Logical Block Application Tag

		ELBATM	15	31:16	Expected Logical Block Application Tag Mask
Reservation Acquire	ReservationAcquire	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		SGL1	6	63:0	SGL Entry 1
		RACQA	10	2:0	Reservation Acquire Action
		IEKEY	10	3	Ignore Existing Key
		RSVD	10	7:4	Reserved
		RTYPE	10	15:8	Reservation Type
		RSVD1	10	31:16	Reserved
Reservation Register	ReservationRegister	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		SGL1	6	63:0	SGL Entry 1
		RREGA	10	2:0	Reservation Register Action
		IEKEY	10	3	Ignore Existing Key
		RSVD	10	29:4	Reserved
		CPTPL	10	31:30	Change Persist Through Power Loss State
Reservation Release	ReservationRelease	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		SGL1	6	63:0	SGL Entry 1
		RRELA	10	2:0	Reservation Release Action
		IEKEY	10	3	Ignore Existing Key
		RSVD	10	7:4	Reserved
		RTYPE	10	15:8	Reservation Type
		RSVD1	10	31:16	Reserved
Reservation Report	ReservationReport	PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		SGL1	6	63:0	SGL Entry 1
		NUMD	10	31:16	Number of Dwords
Write	Write	MPTR	4	63:0	Metadata Pointer
		PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2

		MSGLP	4	63:0	Metadata SGL Segment Pointer
		SGL1	6	127:0	SGL Entry 1
		SLBA	10	63:0	Starting LBA
		NLB	12	15:0	Number of Logical Blocks
		RSVD	12	24:16	Reserved
		PRINFO	12	29:26	Protection Information Field
		FUA	12	30	Force Unit Access
		LR	12	31	Limited Retry
		ACCF	13	3:0	Access Frequency
		ACCL	13	5:4	Access Latency
		SEQR	13	6	Sequential Request
		INCOM	13	7	Incompressible
		RSVD1	13	31:8	Reserved
		ILBRT	14	31:0	Initial Logical Block Reference Tag
		LBAT	15	15:0	Logical Block Application Tag
		LBATM	15	31:16	Logical Block Application Tag Mask
Write Uncorrectable	WriteUncorrectable	SLBA	10	63:0	Starting LBA
		NLB	12	15:0	Number of Logical Blocks
		RSVD	12	31:16	Reserved
Compare	Compare	MPTR	4	63:0	Metadata Pointer
		PRP1	6	63:0	PRP Entry 1
		PRP2	8	63:0	PRP Entry 2
		MSGLP	4	63:0	Metadata SGL Segment Pointer
		SGL1	6	127:0	SGL Entry 1
		SLBA	10	63:0	Starting LBA
		NLB	12	15:0	Number of Logical Blocks
		RSVD	12	25:16	Reserved
		PRINFO	12	29:26	Protection Information Field
		FUA	12	30	Force Unit Access
		LR	12	31	Limited Retry
		EILBRT	14	31:00	Expected Initial Logical Block Reference Tag
		ELBAT	15	15:00	Expected Logical Block Application Tag
		ELBATM	15	31:16	Expected Logical Block Application Tag Mask
Dataset Management	DatasetManagement	MPTR	4	63:0	Metadata Pointer
		PRP1	6	63:0	PRP Entry 1

		PRP2	8	63:0	PRP Entry 2
		MSGLP	4	63:0	Metadata SGL Segment Pointer
		SGL1	6	127:0	SGL Entry 1
		NR	10	7:0	Number of Ranges
		RSVD	10	31:8	Reserved
		IDR	11	0	Attribute - Integral Dataset for Read
		IDW	11	1	Attribute - Integral Dataset for Write
		AD	11	2	Attribute - Deallocate
		RSVD1	11	31:3	Reserved
Write Zeroes	WriteZeros	SLBA	10	63:0	Starting LBA
		NLB	12	15:0	Number of Logical Blocks
		RSVD	12	24:16	Reserved
		PRINFO	12	29:26	Protection Information Field
		FUA	12	30	Force Unit Access
		LR	12	31	Limited Retry
		DSM	13	7:0	Dataset Management
		RSVD1	13	10:8	Reserved
		ILBRT	14	31:0	Initial Logical Block Reference Tag
		LBAT	15	15:0	Logical Block Application Tag
		LBATM	15	31:16	Logical Block Application Tag Mask
Get / Set Features: Arbitration	GetFeatures / SetFeatures	AB	11	2:0	Arbitration Burst
		RSVD	11	7:3	Reserved
		LPW	11	15:8	Low Priority Weight
		MPW	11	23:16	Medium Priority Weight
		HPW	11	31:24	High Priority Weight
Get / Set Features: Power Management	GetFeatures / SetFeatures	PS	11	4:0	Power State
		RSVD	11	31:5	Reserved
Get / Set Features: LBA Range Type	GetFeatures_LBARangeType<index> / SetFeatures_LBARangeType<index>	NUM	11	5:0	Number of LBA Ranges
		RSVD	11	31:6	Reserved
Get / Set Features: Temperature	GetFeatures / SetFeatures	TMPTH	11	15:0	Temperature Threshold
		RSVD	11	31:16	Reserved

Threshold					
Get / Set Features: Error Recovery	GetFeatures / SetFeatures	TLER	11	15:0	Time Limited Error Recovery
		RSVD	11	31:16	Reserved
Get / Set Features: Volatile Write Cache	GetFeatures / SetFeatures	WCE	11	0	Volatile Write Cache Enable
		RSVD	11	31:1	Reserved
Get / Set Features: Number of Queues	GetFeatures / SetFeatures	NSQR	11	15:0	Number of I/O Submission Queues Requested
		NCQR	11	31:16	Number of I/O Completion Queues Requested
Get / Set Features: Interrupt Coalescing	GetFeatures / SetFeatures	THR	11	7:0	Aggregation Threshold
		TIME	11	15:8	Aggregation Time
		RSVD	11	31:16	Reserved
Get / Set Features: Interrupt Vector Configuration	GetFeatures / SetFeatures	IV	11	15:0	Interrupt Vector
		CD	11	16	Coalescing Disable
		RSVD	11	31:17	Reserved
Get / Set Features: Write Atomicity	GetFeatures / SetFeatures	DN	11	0	Disable Normal
		RSVD	11	31:1	Reserved
Get / Set Features: Asynchronous Event Configuration	GetFeatures / SetFeatures	SMART	11	7:0	SMART / Health Critical Warnings
		RSVD	11	31:8	Reserved
Get / Set Features: Autonomous Power State Transition	GetFeatures / SetFeatures	APSTE	11	0	Autonomous Power State Transition Enable
		RSVD	11	31:1	Reserved
Get / Set Features: Software Progress Marker	GetFeatures / SetFeatures	PBSLC	11	7:0	Pre-boot Software Load Count
		RSVD	11	31:8	Reserved
Get / Set Features: Reservation Notification Configuration	GetFeatures / SetFeatures	RSVD	11	0	Reserved
		REGPRE	11	1	Mask Registration Preempted Notification
		RESREL	11	2	Mask Reservation Released Notification
		RESPRE	11	3	Mask Reservation Preempted Notification
		RSVD1	11	31:4	Reserved
Get / Set Features: Reservation Persistence	GetFeatures / SetFeatures	PTPL	11	0	Persist Through Power Loss
		RSVD	11	31:1	Reserved

Completion Queue Entry Data

Command	**Parameter**	**Fields**	**starting**	**Offset**	**Meaning**

			DWORD	in bits	
Asynchronous Event Request	AsyncEventRequest	AE_TYPE	0	2:0	Asynchronous Event Type
		RSVD	0	7:3	Reserved
		AE_INFO	0	15:8	Asynchronous Event Information
		ASSOCIATED_LOG_PAGE	0	23:16	Associated Log Page
		RSVD1	0	31:24	Reserved
Get / Set Features: Number of Queues	GetFeatures / SetFeatures	NSQA	0	15:0	Number of I/O Submission Queues Allocated
		NCQA	0	31:16	Number of I/O Completion Queues Allocated

Payload Data

Command	**Parameter**	**Fields**	**starting DWORD**	**Offset in bits**	**Meaning**
Get / Set Features: LBA Range Type	GetFeatures / SetFeatures	TYPE	0	7:0	Attributes
		ATTRIBUTES_BIT0	0	8	Attributes bit 0
		ATTRIBUTES_BIT1	0	9	Attributes bit 1
		RSVD	0	15:10	Reserved
		RSVD1	0	127:16	Reserved
		SLBA	4	63:0	Starting LBA
		NLB	6	63:0	Number of logical blocks
		GUID	8	127:0	Unique Identifier
		RSVD2	12	127:0	Reserved
Get / Set Features: Autonomous Power State Transition	GetFeatures / SetFeatures	RSVD	0	2:0	Reserved
		ITPS	0	7:3	Idle Transition Power State
		ITPT	0	31:8	Idle Time Prior to Transition
		RSVD1	1	31:0	Reserved
Get / Set Features: Host Identifier	GetFeatures / SetFeatures	HOSTID	0	63:0	Host Identifier
GetLogPage: Error Information	GetLogPage	ERROR_COUNT	0	63:0	Error Count
		SUB_Q_ID	2	15:0	Submission Queue ID
		COMMAND_ID	2	31:16	Command ID
		STATUS_FIELD	3	15:0	Status Field
		BYTE_WITH_ERROR	3	23:16	Parameter Error Location: Byte
		BIT_WITH_ERROR	3	26:24	Parameter Error Location: Bit
		RSVD	3	31:27	Reserved
		LBA	4	63:0	LBA

		NAMESPACE	6	31:0	Namespace
		VENDOR_SPECIFIC_INFO	7	7:0	Vendor Specific Information Available
		RSVD1	7	271:8	Reserved
GetLogPage: SMART / Health Information Log	GetLogPage	CRITICAL_WARNING_BIT0	0	0	Critical Warning Bit 0
		CRITICAL_WARNING_BIT1	0	1	Critical Warning Bit 1
		CRITICAL_WARNING_BIT2	0	2	Critical Warning Bit 2
		CRITICAL_WARNING_BIT3	0	3	Critical Warning Bit 3
		CRITICAL_WARNING_BIT4	0	4	Critical Warning Bit 4
		RSVD	0	7:5	Reserved
		TEMPERATURE	0	23:8	Temperature
		AVAILABLE_SPARE	0	31:24	Available Spare
		AVAILABLE_SPARE_THRESHOLD	1	7:0	Available Spare Threshold
		PERCENTAGE_USED	1	15:8	Percentage Used
		RSVD1	1	223:16	Reserved
		DATA_UNITS_READ	8	127:0	Data Units Read
		DATA_UNITS_WRITTEN	12	127:0	Data Units Written
		HOST_READ_COMMANDS	16	127:0	Host Read Commands
		HOST_WRITE_COMMANDS	20	127:0	Host Write Commands
		CONTROLLER_BUSY_TIME	24	127:0	Controller Busy Time
		POWER_CYCLES	28	127:0	Power Cycles
		POWER_ON_HOURS	32	127:0	Power On Hours
		UNSAFE_SHUTDOWNS	36	127:0	Unsafe Shutdowns
		MEDIA_ERRORS	40	127:0	Media Errors

		NUMBER_OF_ERROR_INFORMATION_LOG_ENTRIES	44	127:0	Number of Error Information Log Entries
		RSVD2	48	2559:0	Reserved
GetLogPage: Firmware Slot Information	GetLogPage	AFI_BITS0_2	0	2:0	Active Firmware Info Bits 2:0
		RSVD	0	3	Reserved
		AFI_BITS4_6	0	6:4	Active Firmware Info Bits 6:4
		RSVD1	0	7	Reserved
		RSVD2	0	63:8	Reserved
		FRS1	2	63:0	Firmware Revision for Slot 1
		FRS2	4	63:0	Firmware Revision for Slot 2
		FRS3	6	63:0	Firmware Revision for Slot 3
		FRS4	8	63:0	Firmware Revision for Slot 4
		FRS5	10	63:0	Firmware Revision for Slot 5
		FRS6	12	63:0	Firmware Revision for Slot 6
		FRS7	14	63:0	Firmware Revision for Slot 7
		RSVD3	16	3583:0	Reserved
GetLogPage: Reservation Notification	GetLogPage	LOG_PAGE_COUNT	0	63:0	Log Page Count
		RN_LOG_PAGE_TYPE	2	7:0	Reservation Notification Log Page Type
		NUMBER_OF_AVAILABLE_LOG_PAGES	2	15:8	Number of Available Log Pages
		RSVD	2	31:16	Reserved
		NAMESPACE_ID	3	31:0	Namespace ID
		RSVD1	4	415:0	Reserved
Identify Controller; Power State Descriptor	Identify_PSD<index>	MP	0	15:0	Maximum Power
		RSVD11	0	23:16	Reserved
		MPS	0	24	Max Power Scale
		NOPS	0	25	Non-Operational State
		RSVD12	0	31:26	Reserved
		ENLAT	1	31:0	Entry Latency
		EXLAT	2	31:0	Exit Latency
		RRT	3	4:0	Relative Read Throughput
		RSVD13	3	7:5	Reserved
		RRL	3	12:8	Relative Read Latency
		RSVD14	3	15:13	Reserved
		RWT	3	20:16	Relative Write Throughput
		RSVD15	3	23:21	RSVD15
		RWL	3	28:24	Relative Write Latency
		RSVD16	3	159:29	RSVD15
Identify	Identify	VID	0	15:0	PCI Vendor ID

Controller		SSVID	0	31:16	PCI Subsystem Vendor ID
		SN	1	159:0	Serial Number
		MN	6	319:0	Model Number
		FR	16	63:0	Firmware Revision
		RAB	18	7:0	Recommended Arbitration Burst
		IEEE	18	31:8	IEEE OUI Identifier
		CMIC_BIT0	19	7:0	Controller Multi-Path I/O and Namespace Sharing Capabilities Bit 0
		CMIC_BIT1	19	8:1	Controller Multi-Path I/O and Namespace Sharing Capabilities Bit 1
		CMIC_BIT2	19	9:2	Controller Multi-Path I/O and Namespace Sharing Capabilities Bit 2
		RSVD	19	42:3	Reserved
		MDTS	19	15:8	Maximum Data Transfer Size
		CNTLID	19	31:16	Controller ID
		RSVD1	20	1407:0	Reserved
		OACS_BIT0	64	0	Optional Admin Command Support Bit 0
		OACS_BIT1	64	1	Optional Admin Command Support Bit 1
		OACS_BIT2	64	2	Optional Admin Command Support Bit 2
		RSVD2	64	15:3	Reserved
		ACL	64	23:16	Abort Command Limit
		AERL	64	31:24	Asynchronous Event Request Limit
		FRMW_BIT0	65	0	Firmware Updates bit 0
		FRMW_BITS1_3	65	3:1	Firmware Updates bits 1:3
		RSVD3	65	7:4	Reserved
		LPA_BIT0	65	8	Log Page Attributes bit 0
		RSVD4	65	15:9	Reserved
		ELPE	65	23:16	Error Log Page Entries
		NPSS	65	31:24	Number of Power States Support
		AVSCC_BIT0	66	0	Admin Vendor Specific Command Configuration bit 0
		RSVD5	66	7:1	Reserved

		APSTA_BIT0	66	8	Autonomous Power State Transition Attributes bit 0
		RSVD4	66	15:9	Reserved
		RSVD5	66	1983:16	Reserved
		SQES_BITS0_3	128	3:0	Submission Queue Entry Size bits 3:0
		SQES_BITS4_7	128	7:4	Submission Queue Entry Size bits 7:4
		CQES_BITS0_3	128	11:8	Completion Queue Entry Size bits 3:0
		CQES_BITS4_7	128	15:12	Completion Queue Entry Size bits 7:4
		RSVD	128	31:16	Reserved
		NN	129	31:0	Number of Namespaces
		ONCS_BIT0	130	7:0	Optional NVM Command Support bit 0
		ONCS_BIT1	130	8:1	Optional NVM Command Support bit 1
		ONCS_BIT2	130	9:2	Optional NVM Command Support bit 2
		ONCS_BIT3	130	10:3	Optional NVM Command Support bit 3
		ONCS_BIT4	130	11:4	Optional NVM Command Support bit 4
		ONCS_BIT5	130	12:5	Optional NVM Command Support bit 5
		RSVD1	130	9:0	Reserved
		FUSES_BIT0	130	16	Fused Operation Support bit 0
		RSVD2	130	31:17	Reserved
		FNA_BIT0	131	0	Format NVM Attributes bit 0
		FNA_BIT1	131	1	Format NVM Attributes bit 1
		FNA_BIT2	131	2	Format NVM Attributes bit 2
		RSVD3	131	7:3	Reserved
		VWC_BIT0	131	8	Volatile Write Cache bit 0
		RSVD4	131	16:10	Reserved
		AWUN	131	31:16	Atomic Write Unit Normal
		AWUPF	132	15:0	Atomic Write Unit Power Fail
		NVSCC_BIT0	132	16	NVM Vendor Specific Command Configuration bit 0
		RSVD5	132	23:17	Reserved
		RSVD6	132	31:24	Reserved

		ACWU	133	15:0	Atomic Compare & Write Unit
		RSVD	133	31:16	Reserved
		SGL_SUPPORT_BIT0	134	0	SGL Support bit 0
		RSVD7	134	15:1	Reserved
		SGL_SUPPORT_BIT16	134	15:1	SGL Support bit 16
		RSVD8	134	30:16	Reserved
		RSVD9	135	1311:0	Reserved
		RSVD10	176	10751:0	Reserved
		VS	768	8191:0	Vendor Specific
Identify Namespace; LBA Format	Identify_LBAF<index>	MS	0	15:0	Metadata Size
		LBADS	0	23:16	LBA Data Size
		RP	0	25:24	Relative Performance
		RSVD8	0	31:26	Reserved
Identify Namespace	Identify	NSZE	0	63:0	Namespace Size
		NCAP	2	63:0	Namespace Capacity
		NUSE	4	63:0	Namespace Utilization
		NSFEAT_BIT0	6	0	Namespace Features bit 0
		RSVD	6	7:1	Reserved
		NLBAF	6	15:8	Number of LBA Formats
		FLBAS_BITS0_3	6	19:16	Formatted LBA Size bits 0 3
		FLBAS_BIT4	6	20	Formatted LBA Size bit 4
		RSVD1	6	23:21	Reserved
		MC_BIT0	6	24	Metadata Capabilities bit 0
		MC_BIT1	6	25	Metadata Capabilities bit 1
		RSVD2	6	31:26	Reserved
		DPC_BIT0	7	0	End-to-end Data Protection Capabilities bit 0
		DPC_BIT1	7	1	End-to-end Data Protection Capabilities bit 1
		DPC_BIT2	7	2	End-to-end Data Protection Capabilities bit 2
		DPC_BIT3	7	3	End-to-end Data Protection Capabilities bit 3
		DPC_BIT4	7	4	End-to-end Data Protection Capabilities bit 4
		RSVD3	7	7:5	Reserved
		DPS_BITS0_2	7	10:8	End-to-end Data Protection Type Settings bits 0 2

		DPS_BIT3	7	11	End-to-end Data Protection Type Settings bit 3
		RSVD4	7	15:12	Reserved
		NMIC_BIT0	7	16	Namespace Multi-path I/O and Namespace Sharing Capabilities bit 0
		RSVD5	7	23:17	Reserved
		RESCAP_BIT 0	7	24	Reservation Capabilities bit 0
		RESCAP_BIT 1	7	25	Reservation Capabilities bit 1
		RESCAP_BIT 2	7	26	Reservation Capabilities bit 2
		RESCAP_BIT 3	7	27	Reservation Capabilities bit 3
		RESCAP_BIT 4	7	28	Reservation Capabilities bit 4
		RESCAP_BIT 5	7	29	Reservation Capabilities bit 5
		RESCAP_BIT 6	7	30	Reservation Capabilities bit 6
		RSVD6	7	31	Reserved
		RSVD7	8	703:0	Reserved
		EUI64	30	63:0	IEEE Extended Unique Identifier
		RSVD9	48	1535:0	Reserved
		VS	96	29695:0	Vendor Specific
Identify List of Namespaces; NSID	Identify_NSID<index>	NSID	0	31:0	Namespace ID
Reservation Acquire	ReservationAcquire	CRKEY	0	63:0	Current Reservation Key
		PRKEY	2	63:0	Preempt Reservation Key
Reservation Register	ReservationRegister	CRKEY	0	63:0	Current Reservation Key
		NRKEY	2	63:0	New Reservation Key
Reservation Release	ReservationRelease	CRKEY	0	63:0	Current Reservation Key
Reservation Report	ReservationReport	GEN	0	31:0	Generation
		RTYPE	1	7:0	Reservation Type
		REGCTL	1	23:8	Number of Registered Controllers
		RSVD	1	39:24	Reserved
		PTPLS	2	15:8	Persist Through Power Loss State
		RSVD1	2	127:16	Reserved
Reservation	ReservationReport	CNTLID	0	15:0	Controller ID

Report; Registered Controller		RCSTS_BIT0	0	16	Reservation Status bit 0
		RSVD	0	23:17	Reserved
		RSVD1	0	63:24	Reserved
		HOSTID	2	63:0	Host Identifier
		RKEY	4	63:0	Reservation Key
DatasetManagement: Range Definition	DatasetManagement_Range<index>	AF	0	3:0	Access Frequency
		AL	0	5:4	Access Latency
		RSVD2	0	7:6	Reserved
		SR	0	8	Sequential Read Range
		SW	0	9	Sequential Write Range
		WP	0	10	Write Prepare
		RSVD3	0	23:11	Reserved
		CAS	0	31:24	Command Access Size
		LEN_LB	1	31:0	Length in logical blocks
		SLBA	2	63:0	Starting LBA

The Feature Identifier field of Get Feature and Set Feature commands can be accessed the following way: **in.GetFeature_FID** and **in.SetFeature_FID** respectively. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Feature Identifier	**Value**	**Name**
_NVMC_FID_ARBITRATION	0x01	Arbitration
_NVMC_FID_POWERMANAGEMENT	0x02	Power Management
_NVMC_FID_LBARANGETYPE	0x03	LBA Range Type
_NVMC_FID_TEMPERATURETHRESHOLD	0x04	Temperature Threshold
_NVMC_FID_ERRORRECOVERY	0x05	Error Recovery
_NVMC_FID_VOLATILEWRITECACHE	0x06	Volatile Write Cache
_NVMC_FID_NUMBEROFQUEUES	0x07	Number of Queues
_NVMC_FID_INTERRUPTCOALESCING	0x08	Interrupt Coalescing
_NVMC_FID_INTERRUPTVECTORCONFIG	0x09	Interrupt Vector Configuration
_NVMC_FID_WRITEATOMICITY	0x0A	Write Atomicity
_NVMC_FID_ASYNCEVENTCONFIG	0x0B	Asynchronous Event Configuration
_NVMC_FID_AUTOPOWERSTATETRANS	0x0C	Autonomous Power State Transition
_NVMC_FID_SOFTPROGRESMARKER	0x80	Software Progress Marker
_NVMC_FID_HOSTIDENTIFIER	0x81	Host Identifier
_NVMC_FID_RESERVNOTIFICMASK	0x82	Reservation Notification Mask
_NVMC_FID_RESERVPERSISTANCE	0x83	Reservation Persistance

The Log Page Identifier field of Get Log Page command can be accessed the following way: **in.GetLogPage_LID**. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Log Page Identifier	**Value**	**Name**
_NVMC_LID_ERROR_INFORMATION	0x01	Error Information
_NVMC_LID_SMART_HEALTH_INFORMATION	0x02	SMART / Health Information
_NVMC_LID_FIRMWARE_SLOT_INFORMATION	0x03	Firmware Slot Information
_NVMC_LID_RESERVATION_NOTIFICATION	0x80	Reservation Notification

The Controller or Namespace Structure of Identify command can be accessed the following way: **in.Identify_CNS**. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Controller or Namespace structure	Value	Name
_NVMC_IDENTIFY_CNS_NAMESPACE	0x00	The Identify Namespace
_NVMC_IDENTIFY_CNS_CONTROLLER	0x01	The Identify Controller
_NVMC_IDENTIFY_CNS_LIST_OF_NAMESPACES	0x02	A list of up to 1024 namespace Ids
_NVMC_IDENTIFY_CNS_RSVD	0x03	Reserved

The Asynchronous Event Type of Asynchronous Event Request command can be accessed the following way: **in. AsyncEventRequest_AE_TYPE**. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Asynchronous Event Type	Value	Name
_NVMC_ASYNC_EVENT_TYPE_ERROR_STATUS	0x00	Error Status
_NVMC_ASYNC_EVENT_TYPE_SMART_HEALTH_STATUS	0x01	SMART / Health Status
_NVMC_ASYNC_EVENT_TYPE_RSVD_FIRST	0x02	Reserved
_NVMC_ASYNC_EVENT_TYPE_RSVD_LAST	0x05	Reserved
_NVMC_ASYNC_EVENT_TYPE_ID_COMMAND_SET_SPECIFIC_STATUS	0x06	I/O Command Set Specific status
_NVMC_ASYNC_EVENT_TYPE_VENDOR_SPECIFIC	0x07	Vendor Specific

5.2.9.1 Metric values

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this NVM Transaction transferred, an integer value.

in.Metric_NumOfNVMTras: Metric presenting the total number of NVM Transactions that compose this NVM Command, an integer value.

in. Metric_LatencyTime: Metric presenting time measured from the end of transmission of the SQ Doorbell to the completion of data delivery, a VSE time object value (see 9.1 VSE Time Object for details)

in. Metric_ResponseTime: Metric presenting time it took to transmit this NVM Command on the link, from the beginning of the first packet to the end of the last packet in the command, a VSE time object value (see 9.1 VSE Time Object for details)

in. Metric_SubmissionDoorbell_CompletionDoorbell_DeltaTime: Metric presenting time measured between Submission Doorbell transaction and Completion Doorbell transaction, a VSE time object value (see 9.1 VSE Time Object for details)

in. Metric_SubmissionDoorbell_CompletionCommand_DeltaTime: Metric presenting time measured between Submission Doorbell transaction and Completion Command transaction, a VSE time object value (see 9.1 VSE Time Object for details)

in. Metric_SubmissionCommand_CompletionCommand_DeltaTime: Metric presenting time measured between Submission Command transaction and Completion Command transaction, a VSE time object value (see 9.1 VSE Time Object for details)

5.2.10 AHCI transaction-specific set of members

Valid for AHCI transactions only. Undefined for other events.

All the AHCI-specific values are present in the input context for AHCI transactions, depending upon the type of register. Also the common PayloadLength and Payload values reflect the total combined payload for the AHCI transaction. In addition to that, the following values exist:

in.AHCIRegId: Contains the numeric encoding of the AHCI register type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Register type	Value	Register description	Corresponding event type
_AHCI_CAP	2	Host Capabilities	_AHCI_HBA_REG
_AHCI_GHC	3	Global Host Control	
_AHCI_IS	4	Interrupt Status	
_AHCI_PI	5	Ports Implemented	
_AHCI_VS	6	Version	
_AHCI_CCC_CTL	7	Command Completion Coalescing Control	
_AHCI_CCC_PORTS	8	Command Completion Coalescing Ports	
_AHCI_EM_LOC	9	Enclosure Management Location	
_AHCI_EM_CTL	10	Enclosure Management Control	
_AHCI_CAP2	11	Host Capabilities Extended	
_AHCI_BOHC	12	BIOS/OS Handoff Control and Status	
_AHCI_RESERVED_HBA	13	Reserved	
_AHCI_RESERVED_NVMHCI	14	Reserved for NVMHCI	
_AHCI_VENDOR_SPECIFIC_REGISTERS	15	Vendor Specific	
_AHCI_PxCLB	16	Port x Command List Base Address	_AHCI_PORT_REG
_AHCI_PxCLBU	17	Port x Command List Base Address Upper 32-Bits	
_AHCI_PxFB	18	Port x FIS Base Address	
_AHCI_PxFBU	19	Port x FIS Base Address Upper 32-Bits	
_AHCI_PxIS	20	Port x Interrupt Status	
_AHCI_PxIE	21	Port x Interrupt Enable	
_AHCI_PxCMD	22	Port x Command and Status	
_AHCI_PxReserved1	23	Reserved1	
_AHCI_PxTFD	24	Port x Task File Data	
_AHCI_PxSIG	25	Port x Signature	

_AHCI_PxSSTS	26	Port x Serial ATA Status (SCR0: SStatus)	
_AHCI_PxSCTL	27	Port x Serial ATA Control (SCR2: SControl)	
_AHCI_PxSERR	28	Port x Serial ATA Error (SCR1: SError)	
_AHCI_PxSACT	29	Port x Serial ATA Active (SCR3: SActive)	
_AHCI_PxCI	30	Port x Command Issue	
_AHCI_PxSNTF	31	Port x Serial ATA Notification (SCR4: SNotification)	
_AHCI_PxFBS	32	Port x FIS-based Switching Control	
_AHCI_PxDEVSLP	33	Port x Device Sleep	
_AHCI_PxReserved2	34	Reserved2	
_AHCI_PxVS	35	Port x Vendor Specific	
_AHCI_COMMAND_HEADER	36	Command header	_AHCI_CMND_LIST
_AHCI_DSFIS	44	DMA Setup FIS	_AHCI_RECEIVED_FIS
_AHCI_PSFIS	45	PIO Setup FIS	
_AHCI_RFIS	46	D2H Register FIS	
_AHCI_SDBFIS	47	Set Device Bits FIS	
_AHCI_UFIS	48	Unknown FIS (up to 64 bytes)	
_AHCI_RF_RESERVED	49	Reserved	
_AHCI_CFIS	50	Command FIS	_AHCI_CMND_TABLE
_AHCI_ACMD	51	ATAPI Command	
_AHCI_CT_RESERVED	52	Reserved	
_AHCI_PRDT	53	Physical Region Descriptor Table	
_AHCI_DATA	58	Actual data pointed by DBA && DBAU	

in.AHCIRegIdAsString: Contains an AHCI register name.

in.PortNum: Contains a port number.

in.SlotNum: Contains a slot number.

in.AHCITraHasError: If set to a non-zero value, indicates AHCI transaction has errors.

in.AHCIErrorId: Contains the numeric encoding of the AHCI error type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Error type	Value	Error name	Error description
_AHCI_ERROR_RESERVED_NOT_NULL	0	Reserved field is not 0	Reserved registers and fields must be 0 for AHCI
_AHCI_ERROR_INCOMPLETE_TRA	1	Incomplete	Incomplete AHCI

			transaction
_AHCI_ERROR_LOGICAL_ERROR	2	Logical error	Discrepancy between the states of different registers, wrong order of transactions, etc.
_AHCI_ERROR_RO_VIOLATION	3	Read-only field changed	An attempt of writing to the read-only fields and registers
_AHCI_ERROR_COMPLETER_ABORT	4	Completer abort	Completer abort transaction
_AHCI_ERROR_TRA_STARTS_FROM_MIDDLE	5	Unexpected register offset	Incomplete AHCI transaction, that starts from the middle of register
_AHCI_ERROR_REG_ACCESS_VIOLATION	6	Register access violation	HBA register access is longer than 64 bits or crosses 8-byte alignment boundary
_AHCI_ERROR_TABLE_ALIGNMENT_ERROR	7	Address is not properly aligned	Table base address is not properly aligned in memory
_AHCI_ERROR_INVALID_HBA_STATE	8	Invalid HBA state	An attempt to do something forbidden in current HBA state.
_AHCI_ERROR_INVALID_FIS_SIZE	9	Invalid FIS size	FIS size doesn't correspond to spec or CFIS.CFL field.
_AHCI_ERROR_UNKNOWN	10	Unknown error	All other errors.
_AHCI_ERROR_NO_ERROR	12	No errors	Correct transaction

in.AHCIErrorIdAsString: Contains an AHCI error name

The following table shows the list of AHCI registers and their fields defined in the input context. The fields can be accessed by using "_". E.g. **in.CAP_SXS** contains the numeric encoding of the SXS field of CAP register.

Note: If length of returned value is bigger than 1 dword, please, specify dword by using “_DW” and dword index, otherwise string in a hex format will be returned. For example **in.HBA_VENDOR_SPECIFIC_DW3** contains the numeric encoding of the 3rd dword of HBA Vendor Specific field.

Parameter	**Fields**	**Offset**	**Meaning**
CAP	NP	04:00	Number of Ports
	SXS	05	Supports External SATA
	EMS	06	Enclosure Management Supported
	CCCS	07	Command Completion Coalescing Supported
	NCS	12:08	Number of Command Slots
	PSC	13	Partial State Capable
	SSC	14	Slumber State Capable
	PMD	15	PIO Multiple DRQ Block (PMD)
	FBSS	16	FIS-based Switching Supported
	SPM	17	Supports Port Multiplier
	SAM	18	Supports AHCI mode only

	RSVD	19	Reserved
	ISS	23:20	Interface Speed Support
	SCLO	24	Supports Command List Override
	SAL	25	Supports Activity LED
	SALP	26	Supports Aggressive Link Power Management
	SSS	27	Supports Staggered Spin-up
	SMPS	28	Supports Mechanical Presence Switch
	SSNTF	29	Supports SNotification Register
	SNCQ	30	Supports Native Command Queuing
	S64A	31	Supports 64-bit Addressing
GHC	HR	00	HBA Reset
	IE	01	Interrupt Enable
	MRSM	02	MSI Revert to Single Message
	RSVD	30:03	Reserved
	AE	31	AHCI Enable
IS	IPS	31:00	Interrupt Pending Status
PI	PI	31:00	Port Implemented
VS	MNR	15:00	Minor Version Number
	MJR	31:16	Major Version Number
CCC_CTL	EN	0	Enable
	RSVD	2:1	Reserved
	INT	7:3	Interrupt
	CC	15:8	Command Completions
	TV	31:16	Timeout Value
CCC_PORTS	PRT	31:0	Ports
EM_LOC	SZ	15:0	Buffer Size
	OFST	31:16	Offset
EM_CTL	STS_MR	0	Message Received
	RSVD4	07:01	Reserved
	CTL_TM	8	Transmit Message
	CTL_RST	9	Reset
	RSVD3	15:10	Reserved
	SUPP_LED	16	LED Message Types
	SUPP_SAFTE	17	SAF-TE Enclosure Management Messages
	SUPP_SES2	18	SES-2 Enclosure Management Messages
	SUPP_SGPIO	19	SGPIO Enclosure Management Messages
	RSVD2	23:20	Reserved
	ATTR_SMB	24	Single Message Buffer
	ATTR_XMT	25	Transmit Only

	ATTR_ALHD	26	Activity LED Hardware Driven
	ATTR_PM	27	Port Multiplier Support
	RSVD	31:28	Reserved
CAP2	BOH	00	BIOS/OS Handoff
	NVMP	01	NVMHCI Present
	APST	02	Automatic Partial to Slumber Transitions
	SDS	03	Supports Device Sleep
	SADM	04	Supports Aggressive Device Sleep Management
	DESO	05	DevSleep Entrance from Slumber Only
	RSVD	31:06	Reserved
BOHC	BOS	00	BIOS Owned Semaphore
	OOS	01	OS Owned Semaphore
	SOOE	02	SMI on OS Ownership Change Enable
	OOC	03	OS Ownership Change
	BB	04	BIOS Busy
	RSVD	31:05	Reserved
RSVD_HBA	-		Reserved
RSVD_NVMHCI	-		Reserved for NVMHCI
HBA_VENDOR_SPECIFIC	-		Venfor specific
		Port registers	
PxCLB	RSVD	09:00	Reserved
	CLB	31:10	Command List Base Address
PxCLBU	CLBU	31:00	Command List Base Address Upper
PxFB	RSVD	07:00	Reserved
	FB	31:08	FIS Base Address
PxFBU	FBU	31:00	FIS Base Address Upper
PxIS	DHRS	00	Device to Host Register FIS Interrupt
	PSS	01	PIO Setup FIS Interrupt
	DSS	02	DMA Setup FIS Interrupt
	SDBS	03	Set Device Bits Interrupt
	UFS	04	Unknown FIS Interrupt
	DPS	05	Descriptor Processed
	PCS	06	Port Connect Change Status
	DMPS	07	Device Mechanical Presence Status
	RSVD2	21:08	Reserved
	PRCS	22	PhyRdy Change Status
	IPMS	23	Incorrect Port Multiplier Status
	OFS	24	Overflow Status
	RSVD1	25	Reserved

	INFS	26	Interface Non-fatal Error Status
	IFS	27	Interface Fatal Error Status
	HBDS	28	Host Bus Data Error Status
	HBFS	29	Host Bus Fatal Error Status
	TFES	30	Task File Error Status
	CPDS	31	Cold Port Detect Status
PxIE	DHRE	00	Device to Host Register FIS Interrupt Enable
	PSE	01	PIO Setup FIS Interrupt Enable
	DSE	02	DMA Setup FIS Interrupt Enable
	SDBE	03	Set Device Bits FIS Interrupt Enable
	UFE	04	Unknown FIS Interrupt Enable
	DPE	05	Descriptor Processed Interrupt Enable
	PCE	06	Port Change Interrupt Enable
	DMPE	07	Device Mechanical Presence Enable
	RSVD2	21:08	Reserved
	PRCE	22	PhyRdy Change Interrupt Enable
	IPME	23	Incorrect Port Multiplier Enable
	OFE	24	Overflow Enable
	RSVD	25	Reserved
	INFE	26	Interface Non-fatal Error Enable
	IFE	27	Interface Fatal Error Enable
	HBDE	28	Host Bus Data Error Enable
	HBFE	29	Host Bus Fatal Error Enable
	TFEE	30	Task File Error Enable
	CPDE	31	Cold Presence Detect Enable
PxCMD	ST	00	Start
	SUD	01	Spin-Up Device
	POD	02	Power On Device
	CLO	03	Command List Override
	FRE	04	FIS Receive Enable
	RSVD	07:05	Reserved
	CSS	12:08	Current Command Slot
	MPSS	13	Mechanical Presence Switch State
	FR	14	FIS Receive Running
	CR	15	Command List Running
	CPS	16	Cold Presence State
	PMA	17	Port Multiplier Attached
	HPCP	18	Hot Plug Capable Port
	MPSP	19	Mechanical Presence Switch Attached to Port

	CPD	20	Cold Presence Detection
	ESP	21	External SATA Port
	FBSCP	22	FIS-based Switching Capable Port
	APSTE	23	Automatic Partial to Slumber Transitions Enabled
	ATAPI	24	Device is ATAPI
	DLAE	25	Drive LED on ATAPI Enable
	ALPE	26	Aggressive Link Power Management Enable
	ASP	27	Aggressive Slumber / Partial
	ICC	31:28	Interface Communication Control
PxRSVD	-		Reserved
PxTFD	STS_ERR	00	Error during the transfer.
	STS_CS1	02:01	Command specific
	STS_DRQ	03	Data transfer is requested
	STS_CS2	06:04	Command specific
	STS_BSY	07	Interface is busy
	ERR	15:08	Error
	RSVD	31:16	Reserved
PxSIG	SC	07:00	Sector Count Register
	LBA_LOW	15:08	LBA Low Register
	LBA_MID	23:16	LBA Mid Register
	LBA_HIGH	31:24	LBA High Register
PxSSTS	DET	03:00	Device Detection
	SPD	07:04	Current Interface Speed
	IPM	11:08	Interface Power Management
	RSVD	31:12	Reserved
PxSCTL	DET	03:00	Device Detection Initialization
	SPD	07:04	Speed Allowed
	IPM	11:08	Interface Power Management Transitions Allowed
	SPM	15:12	Select Power Management
	PMP	19:16	Port Multiplier Port
	RSVD	31:20	Reserved
PxSERR	ERR_I	00	Recovered Data Integrity Error
	ERR_M	01	Recovered Communications Error
	ERR_RSVD2	07:02	Reserved
	ERR_T	08	Transient Data Integrity Error
	ERR_C	09	Persistent Communication or Data Integrity Error
	ERR_P	10	Protocol Error
	ERR_E	11	Internal Error
	ERR_RSVD	15:12	Reserved

	DIAG_N	16	PhyRdy Change
	DIAG_I	17	Phy Internal Error
	DIAG_W	18	Comm Wake
	DIAG_B	19	10B to 8B Decode Error
	DIAG_D	20	Disparity Error
	DIAG_C	21	CRC Error
	DIAG_H	22	Handshake Error
	DIAG_S	23	Link Sequence Error
	DIAG_T	24	Transport state transition error
	DIAG_F	25	Unknown FIS Type
	DIAG_X	26	Exchanged
	DIAG_RSVD	31:27	Reserved
PxSACT	DS	31:00	Device Status
PxCI	CI	31:00	Commands Issued
PxSNTF	PMN	15:00	PM Notify
	RSVD	31:16	Reserved
PxFBS	EN	00	Enable
	DEC	01	Device Error Clear
	SDE	02	Single Device Error
	RSVD2	07:03	Reserved
	DEV	11:08	Device To Issue
	ADO	15:12	Active Device Optimization
	DWE	19:16	Device With Error
	RSVD	31:20	Reserved
PxDEVSLP	ADSE	00	Aggressive Device Sleep Enable
	DSP	01	Device Sleep Present
	DETO	09:02	Device Sleep Exit Timeout
	MDAT	14:10	Minimum Device Sleep Assertion Time
	DITO	24:15	Device Sleep Idle Timeout
	DM	28:25	DITO Multiplier
	RSVD	31:29	Reserved
PxRSVD2	-		Reserved
PxVS	-		Vendor Specific
FIS registers			
RFIS	TYPE	07:00	FIS Type
	PMP	11:08	The Port Multiplier Port
	RSVD4	13:12	Reserved
	I	14	Interrupt bit
	RSVD3	15	Reserved

	STATUS	23:16	Status
	ERROR	31:24	Error - Contains the new value of the Er
	LBA_LO	39:32	LBA(7:0)
	LBA_MID	47:40	LBA(15:8)
	LBA_HI	55:48	LBA(23:16)
	DEVICE	63:56	Device
	LBA_LO_EXP	71:64	LBA(31:24)
	LBA_MID_EXP	79:27	LBA(39:32)
	LBA_HI_EXP	87:80	LBA(47:40)
	RSVD2	95:88	Reserved
	COUNT	103:96	Count(7:0)
	COUNT_EXP	111:104	Count(15:8)
	RSVD	127:112	Reserved
DSFIS	TYPE	07:00	FIS Type
	PMP	11:08	The Port Multiplier Port
	RSVD4	12	Reserved
	D	13	Direction
	I	14	Interrupt
	A	15	Auto-Activate
	RSVD3	31:16	Reserved
	DMA_BI_LO	63:32	DMA Buffer Identifier Low
	DMA_BI_HI	95:64	DMA Buffer Identifier High
	RSVD2	127:96	Reserved
	BO	159:128	DMA Buffer Offset
	TC	191:160	DMA Transfer Count
	RSVD	223:192	Reserved
PSFIS	TYPE	07:00	FIS Type
	PMP	11:08	The Port Multiplier Port
	RSVD5	12	Reserved
	D	13	Direction
	I	14	Interrupt
	RSVD4	15	Reserved
	STATUS	23:16	Status
	ERROR	31:24	Error
	LBA_LO	39:32	LBA(7:0)
	LBA_MID	47:40	LBA(15:8)
	LBA_HI	55:48	LBA(23:16)
	DEVICE	63:56	Device
	LBA_MID_EXP	71:64	LBA(39:32)

	LBA_HI_EXP	79:72	LBA(47:40)
	RSVD3	87:80	Reserved
	COUNT	95:88	Count(7:0)
	COUNT_EXP	103:96	Count(15:8)
	RSVD2	111:104	Reserved
	E_STATUS	119:112	E_Status
	TC	135:120	Transfer Count
	RSVD	151:136	Reserved
SDBFIS	TYPE	07:00	FIS Type
	PMP	11:08	The Port Multiplier Port
	RSVD3	13:12	Reserved
	I	14	Interrupt Bit
	N	15	Notification Bit
	STATUS_LO	18:16	Status-Lo
	RSVD2	19	Reserved
	STATUS_HI	22:20	Status-Hi
	RSVD	23	Reserved
	ERROR	31:24	Error
	PROT_SPEC	64:32	Protocol Specific
Command Table			
PRDT	DBA_RSVD	00	Reserved
	DBA	31:01	Data Base Address
	DBAU	63:32	Data Base Address Upper 32-bits
	RSVD	95:64	Reserved
	DI_DBC	117:96	Data Byte Count
	DI_RSVD	126:118	Reserved
	DI_I	127	Description Information: Interrupt on Completion
ACMD	ACMD	-	Description Information: ATAPI Command
Command List			
COMMAND_HEADER	CFL	04:00	Command FIS Length
	A	05	ATAPI
	W	06	Write
	P	07	Prefetchable
	R	08	Reset
	B	09	BIST
	C	10	Clear Busy upon R_OK
	RSVD	11	Reserved
	PMP	15:12	Port Multiplier Port
	PRDTL	31:16	Physical Region Descriptor Table Length

	PRDBC	63:32	Physical Region Descriptor Byte Count
	CTBA_RSVD	70:64	Reserved
	CTBA	95:71	Command Table Descriptor Base Address
	CTBAU	127:96	Command Table Descriptor Base Address Upper 32-bits
	RSVD1	159:128	Reserved
	RSVD2	191:160	Reserved
	RSVD3	223:192	Reserved
	RSVD4	255:224	Reserved

5.2.10.1 Metric values

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this NVM Transaction transferred, an integer value.

in.Metric_NumOfLinkAndSplitTras: Metric presenting the total number of Link and Split Transactions that compose this AHCI Transaction, an integer value.

in. Metric_ResponseTime: Metric presenting time it took to transmit this AHCI Transaction on the link, from the beginning of the first packet to the end of the last packet in the transaction, a VSE time object value (see 9.1 VSE Time Object for details)

5.2.11 ATA transaction-specific set of members

Valid for ATA transactions only. Undefined for other events.

All the ATA-specific values are present in the input context for ATA transactions, depending upon the type of register. Also the common PayloadLength and Payload values reflect the total combined payload for the ATA transaction. In addition to that, the following values exist:

in.ataCommandCode: Contains ATA command code value.

in.ataCommand: Contains the numeric encoding of the ATA command id. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Constant	Value
_ATA_NOP	0
_ATA_RESERVED	1
_ATA_CFA_REQUEST_EXTENDED_ERROR	2
_ATA_DATA_SET_MANAGEMENT	3
_ATA_DEVICE_RESET	4
_ATA_REQUEST_SENSE_DATA_EXT	5
_ATA_OBSOLETE	6
_ATA_RETIRED	7
_ATA_READ_SECTORS	8
_ATA_READ_SECTORS_EXT	9
_ATA_READ_DMA_EXT	10
_ATA_READ_NATIVE_MAX_ADDRESS_EXT	11
_ATA_READ_MULTIPLE_EXT	12
_ATA_READ_STREAM_DMA_EXT	13
_ATA_READ_STREAM_EXT	14
_ATA_READ_LOG_EXT	15
_ATA_WRITE_SECTORS	16
_ATA_WRITE_SECTORS_EXT	17
_ATA_WRITE_DMA_EXT	18
_ATA_SET_MAX_ADDRESS_EXT	19
_ATA_CFA_WRITE_SECTORS_WITHOUT_ERASE	20
_ATA_WRITE_SECTORS	21
_ATA_WRITE_SECTORS_EXT	22
_ATA_WRITE_STREAM_EXT	23
_ATA_WRITE_DMA_FUA_EXT	24
_ATA_WRITE_LOG_EXT	25
_ATA_READ_VERIFY_SECTORS	26
_ATA_READ_VERIFY_SECTORS_EXT	27
_ATA_WRITE_UNCORRECTABLE_EXT	28

_ATA_READ_LOG_DMA_EXT	29
_ATA_CONFIGURE_STREAM	30
_ATA_WRITE_LOG_DMA_EXT	31
_ATA_TRUSTED_NON_DATA	32
_ATA_TRUSTED_RECEIVE	33
_ATA_TRUSTED_RECEIVE_DMA	34
_ATA_TRUSTED_SEND	35
_ATA_TRUSTED_SEND_DMA	36
_ATA_READ_FPDMA_QUEUED	37
_ATA_WRITE_FPDMA_QUEUED	38
_ATA_VENDOR_SPECIFIC	39
_ATA_CFA_TRANSLATE_SECTOR	40
_ATA_EXECUTE_DEVICE_DIAGNOSTIC	41
_ATA_DOWNLOAD_MICROCODE	42
_ATA_DOWNLOAD_MICROCODE_DMA	43
_ATA_PACKET	44
_ATA_IDENTIFY_PACKET_DEVICE	45
_ATA_SMART	46
_ATA_DEVICE_CONFIGURATION_OVERLAY	47
_ATA_SANITIZE_DEVICE	48
_ATA_NV_CACHE	49
_ATA_RESERVED_FOR_THE_COMPACTFLASH_ASSOCIATION	50
_ATA_CFA_ERASE_SECTORS	51
_ATA_READ_MULTIPLE	52
_ATA_WRITE_MULTIPLE	53
_ATA_SET_MULTIPLE_MODE	54
_ATA_READ_DMA	55
_ATA_WRITE_DMA	56
_ATA_CFA_WRITE_MULTIPLE_WITHOUT_ERASE	57
_ATA_WRITE_MULTIPLE_FUA_EXT	58
_ATA_CHECK_MEDIA_CARD_TYPE	59
_ATA_RESERVED_FOR_THE_MEDIA_CARD_PASS_THROUGH_COMMAND_FEATURE_SET	60
_ATA_STANDBY_IMMEDIATE	61
_ATA_IDLE_IMMEDIATE	62
_ATA_STANDBY	63
_ATA_IDLE	64
_ATA_READ_BUFFER	65
_ATA_CHECK_POWER_MODE	66

_ATA_SLEEP	67
_ATA_FLUSH_CACHE	68
_ATA_WRITE_BUFFER	69
_ATA_READ_BUFFER_DMA	70
_ATA_FLUSH_CACHE_EXT	71
_ATA_WRITE_BUFFER_DMA	72
_ATA_IDENTIFY_DEVICE	73
_ATA_SET_FEATURES	74
_ATA_SECURITY_SET_PASSWORD	75
_ATA_SECURITY_UNLOCK	76
_ATA_SECURITY_ERASE_PREPARE	77
_ATA_SECURITY_ERASE_UNIT	78
_ATA_SECURITY_FREEZE_LOCK	79
_ATA_SECURITY_DISABLE_PASSWORD	80
_ATA_READ_NATIVE_MAX_ADDRESS	81
_ATA_SET_MAX_ADDRESS	82

in.ataProtocol: Contains the numeric encoding of the ATA protocol id. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Constant	**Value**	**Description**
_ATA_PROTOCOL_NONE	0	Protocol undefined
_ATA_PROTOCOL_ND	0x1	Non-Data command
_ATA_PROTOCOL_PI	0x2	PIO Data-In command
_ATA_PROTOCOL_PO	0x4	PIO Data-Out command
_ATA_PROTOCOL_DM	0x8	DMA command
_ATA_PROTOCOL_DMQ	0x10	DMA QUEUED command
_ATA_PROTOCOL_DR	0x20	DEVICE RESET command
_ATA_PROTOCOL_DD	0x40	EXECUTE DEVICE DIAGNOSTIC command
_ATA_PROTOCOL_P	0x80	PACKET command
_ATA_PROTOCOL_VS	0x100	Vendor specific

in.ataTraHasError: If set to a non-zero value, indicates ATA command has errors.

in.ataErrorId: Contains the numeric encoding of the ATA error type. The following possible values are defined by VSE and the corresponding constants can be used by scripts:

Error type	**Value**	**Error name**	**Error description**
_ATA_ERROR_WRONG_SUB_TRA	0	Error in Sub Transaction	ATA command contains sub-transaction with error in it (for example AHCI)
_ATA_ERROR_INCOMPLETE_SUB_TRA	1	Incomplete Sub Transaction	ATA command contains incomplete sub-transaction (for example AHCI)

_ATA_ERROR_INCOMPLETE_TRA	2	Incomplete transaction	Incomplete ATA transaction
_ATA_ERROR_LOGICAL_ERROR	3	Logical error	Wrong command order, wrong amount of data transferred, etc.
_ATA_ERROR_ERROR_BIT_SET	4	Error bit set	Command has error bit set to 1
_ATA_ERROR_SHADOW_ERROR	5	Shadow error bit set	Shadow error byte in AHCI is set
_ATA_ERROR_NO_ERROR	7	No error	Correct transaction

in.ATAErrorIdAsString: Contains an ATA error name

in.ataPort: Contains port number.

in.ataSlot: Contains slot number.

in.ataFeatures: Contains feature sector count.

in.ataCount: Contains the numeric encoding of the Sector Count register of the Shadow Register Block.

in.ataLBA: Contains the numeric encoding of the LBA low, mid and high registers of the Shadow Register Block

in.ataLBAEXT: Contains the numeric encoding of the expanded LBA register of the Shadow Register Block.

in.ataDevice: Contains the first byte of the Device register of the Shadow Register Block.

in.ataDevice0: Returns first bit of in.ataDevice value.

in.ataDevice7: Returns last bit of in.ataDevice value.

in.ataOutputStatus: Contains the numeric encoding of transaction status register.

in.ataBSY: If set to a non-zero value, indicates that the device is accessing the registers Indicates the interface is busy.

in.ataDRDY: If set to a non-zero value, indicates that the device is capable of responding to a command.

in.ataDF: If set to a non-zero value, indicates that the device has detected a write fault condition.

in.ataDSC: If set to a non-zero value, indicates that a seek has been completed and the device head is settled over a track.

in.ataDRQ:Data If set to a non-zero value, indicates that the device is ready to transfer a word or byte of data between the host and the device.

in.ataCORR: If set to a non-zero value, indicates that a correctable data error was encountered and the data has been corrected. This condition does not terminate a data transfer.

in.ataIDX: Is set to a non-zero value once per revolution.

in.ataERR: If set to a non-zero value, indicates an error during the transfer

in.ataPRIO: If set to a non-zero value, then the command shall be high priority. Otherwise, the command shall be normal priority.

in.ataNCQ: Contains the numeric encoding of the NCQ Tag field.

in.ataRARC: Returns the RARC bit

in.ataICC: Contains the numeric encoding of the Isochronous Command Completion field.

in.ataHybridInfo: If set to a non-zero value, then the device supports the hybrid information feature.

in.ataPayloadLength: Contains the exact payload size (transferred data) in bytes.

in.ataRequestedBytesCount: Contains the requested data size in bytes

in.ataHasData: If set to a non-zero value, indicates payload presence.

in.ataIsDeviceToHostTransition: If set to a non-zero value, indicates the Host to Device direction.

in.ataHasErrors: If set to a non-zero value, indicates an error in decoded transaction.

in.ataContainsPxSACT: If set to a non-zero value, indicates PxSACT register usage

in.ataContainsCFIS: If set to a non-zero value, indicates CFIS register usage

in.ataContainsSDBFIS: If set to a non-zero value, indicates SDBFIS register usage

in.ataContainsPSFIS: If set to a non-zero value, indicates PSFIS register usage

in.ataContainsRFIS: If set to a non-zero value, indicates RFIS register usage

in.ataContainsInterruptD2H: If set to a non-zero value, indicates Device-To-Host interrupt happened

in.ataContainsInterruptH2D: If set to a non-zero value, indicates Host-To-Device interrupt happened

in.ataContainsIncompleteSubTra: If set to a non-zero value, indicates ATA transaction has incomplete sub-transactions

in.ataContainsErrorSubTra: If set to a non-zero value, indicates ATA transaction has sub-transactions with errors

in.ataHasShadowRegister: If set to a non-zero value, indicates a RFIS Shadow Register presence

in.ataRFISShadowRegister: Contains the numeric encoding of the RFIS Shadow Register

5.2.11.1 Metric values

in.Metric_Throughput: Metric presenting transaction payload divided by response time, expressed in **kilobytes** per second, an integer value

in.Metric_PayloadBytes: Metric presenting number of data payload bytes this NVM Transaction transferred, an integer value.

in.Metric_NumOfAHCITras: Metric presenting the total number of AHCI Transactions that compose this ATA Transaction, an integer value.

in. Metric_ResponseTime: Metric presenting time it took to transmit this ATA Transaction on the link, from the beginning of the first packet to the end of the last packet in the transaction, a VSE time object value (see 9.1 VSE Time Object for details)

6 Verification Script Engine Output Context Members

All verification scripts have output contexts – some special structures whose members are filled by the script and can be used inside of the application. (For more details about output contexts, please refer to the *CATC Script Language(CSL) Manual.*) The verification script output contexts have only one member:

out.Result: Result of the whole verification program defined in the verification script.

This member is supposed to have the values:
_VERIFICATION_PROGRESS (set by default when script starts running)
_VERIFICATION_PASSED
_VERIFICATION_FAILED

The last two values should be set if you decide that the recorded trace does (or does not) satisfy the imposed verification conditions. In both cases, the verification script stops running.

If you don't specify any of those values, the result of script execution is set as **_VERIFICATION_FAILED** at exit.

Note: If you don’t care about the results of the script that’s running, please call function ScriptForDisplayOnly() one time before stopping the script. Then the results are **DONE.**

7 Verification Script Engine Events

VSE defines a large group of trace “events” – on packet, link, split, AHCI, ATA, NVM transaction and NVM command levels – that can be passed to a verification script for evaluation or retrieving and displaying some contained information. The information about the type of event can be seen in **in.TraceEvent**. Please refer to the topic "Sending Functions" in this manual for details about how to specify transaction levels and which events should be sent to verification scripts.

7.1 Packet level events

The table below describes the current list of Packet level events (transaction level: 0) and value of **in.TraceEvent:**

Types of Packets	**in.TraceEvent**
Data Link Layer Packets (DLLP)	_PKT_DLLP
Transaction Layer Packets (TLP)	_PKT_TLP
Ordered Sets	_PKT_ORDERED_SET
Link Conditions	_PKT_LINK_CONDITION

7.2 Link Transaction level events

The table below describes the current list of Link Transaction events (transaction level: 1) and value of **in.TraceEvent**:

Types of Link Transactions	**in.TraceEvent**
Memory transactions	_LINK_MEMORY
IO transactions	_LINK_IO
Configuration transactions	_LINK_CONFIG
Message transactions	_LINK_MESSAGE
Completion transactions	_LINK_COMPLETION

7.3 Split Transaction level events

The table below describes the current list of Split Transaction events (transaction level: 2) and value of **in.TraceEvent:**

Types of Split Transactions	**in.TraceEvent**
Memory transactions	_SPLIT_MEMORY
IO transactions	_SPLIT_IO
Configuration transactions	_SPLIT_CONFIG

7.4 NVM Transaction level events

NVME level introduces own events. The table below describes the current list of NVM Transaction events (transaction level: 3) and value of **in.TraceEvent:**

Types of NVM Transactions	**in.TraceEvent**
Transactions that read or write controller register	_NVME_CONTROLLER_REG

Transactions that write doorbell register.	_NVME_DOORBELL_REG
Transactions that transfer admin submission command	_NVME_ADMIN_SUBMISSION_CMD
Transactions that transfer NVM submission command	_NVME_NVM_SUBMISSION_CMD
Transactions that transfer NVM or admin completion command	_NVME_COMPLETION_CMD
Transactions that target idx/dat registers (registers that are at fixed offset from BAR2)	_NVME_IDX_DAT_REG
Transactions that target data which is referenced by PRP or SGL	_NVME_TRANSFERED_DATA
Generated for all PRP list transactions	_NVME_PRP
Generated for all SGL descriptors transactions	_NVME_SGL

7.5 NVM Command level events

NVM Command level introduces own events. The table below describes the current list of NVM Command events (transaction level:) and value of **in.TraceEvent**

Types of NVM Commands	**in.TraceEvent**
Admin command	_NVMC_ADMIN_COMMAND
NVM command	_NVMC_NVM_COMMAND
Security command	_NVMC_SECURITY_COMMAND
Flush command	_NVMC_FLUSH
Write command	_NVMC_WRITE
Read command	_NVMC_READ
Write Uncorrectable command	_NVMC_WRITE_UNCORRECTABLE
Compare command	_NVMC_COMPARE
Write Zeros command	_NVMC_WRITE_ZEROES
Dataset Management command	_NVMC_DATASET_MGMT
Reservation Register command	_NVMC_RESERVATION_REGISTER
Reservation Report command	_NVMC_RESERVATION_REPORT
Reservation Acquire command	_NVMC_RESERVATION_ACQUIRE
Reservation Release command	_NVMC_RESERVATION_RELEASE
Delete I/O Submission Queue command	_NVMC_DELETE_IO_SQ
Create I/O Submission Queue command	_NVMC_CREATE_IO_SQ
Get Log Page command	_NVMC_GET_LOG_PAGE
Delete I/O Completion Queue command	_NVMC_DELETE_IO_CQ
Create I/O Completion Queue command	_NVMC_CREATE_IO_CQ
Identify command	_NVMC_IDENTIFY
Abort command	_NVMC_ABORT
Set Feature command	_NVMC_SET_FEATURE

Get Feature command	_NVMC_GET_FEATURE
Asynchronous Event Request command	_NVMC_ASYNC_EVENT_REQUEST
Firmware Activate command	_NVMC_FIRMWARE_ACTIVATE
Firmware Image Download command	_NVMC_FIRMWARE_IMG_DOWNLOAD
Format NVM command	_NVMC_FORMAT_NVM
Security Send command	_NVMC_SECURITY_SEND
Security Receive command	_NVMC_SECURITY_RECEIVE

7.6 AHCI Transaction level events

The table below describes the current list of AHCI Transaction events (transaction level: 5) and value of **in.TraceEvent:**

Types of AHCI Transactions	**in.TraceEvent**
Transactions that use HBA registers	_AHCI_HBA_REG
Transactions that use Ports registers	_AHCI_PORT_REG
Transactions that use Command list registers	_AHCI_CMND_LIST
Transaction uses FIS registers	_AHCI_RECEIVED_FIS
Transaction uses Command table registers	_AHCI_CMND_TABLE

7.7 ATA Transaction level events

ATA level introduces no events. Use SendAllTraceEvents() function to get ATA events.

8 Sending Functions

This topic contains information about the special group of VSE functions designed to specify which events the verification script should expect to receive.

8.1 SendLevel()

This function specifies that events of the specified transaction level should be sent to the script.

Format: **SendLevel(level)**

Parameters:

`level` Can have one of following values:

_PACKET	(value 0) Send Packet level events
_LINK	(value 1) Send Link Transaction level events
_SPLIT	(value 2) Send Split Transaction level events
_NVME	(value 3) Send NVM Transaction level events
_NVMC	(value 9) Send NVM Command level events
_AHCI	(value 5) Send AHCI Transaction level events
_ATA	(value 6) Send ATA Transaction level events

Example:

```
…
SendLevel( _PACKET); # Send packet level events
```

Remark:

If no level was specified, events of Packet level are sent to the script by default.

8.2 SendLevelOnly()

This function specifies that ONLY events of the specified transaction level should be sent to the script.

Format: SendLevelOnly(level)

Parameters:

`level` Can have one of following values:

_PACKET	(value 0) Send Packet level events
_LINK	(value 1) Send Link Transaction level events
_SPLIT	(value 2) Send Split Transaction level events
_NVME	(value 3) Send NVM Transaction level events
_NVMC	(value 9) Send NVM Command level events
_AHCI	(value 5) Send AHCI Transaction level events
_ATA	(value 6) Send ATA Transaction level events

Example:

```
…
SendLevelOnly( _PACKET ); # Send ONLY packet level events
```

8.3 DontSendLevel()

This function specifies that events of the specified transaction level should NOT be sent to the script.

Format: DontSendLevel(level)

Parameters:

`level` Can have one of following values:

_PACKET	(value 0) Do not send Packet level events
_LINK	(value 1) Do not send Link Transaction level events
_SPLIT	(value 2) Do not send Split Transaction level events
_NVME	(value 3) Do not send NVM Transaction level events
_NVMC	(value 9) Do not send NVM Command level events
_AHCI	(value 5) Do not send AHCI Transaction level events
_ATA	(value 6) Do not send ATA Transaction level events

Example:

```
…
DontSendLevel( _LINK); # DO NOT send link transaction level events
```

8.4 SendChannel()

This function specifies that events that have occurred on the specified channel should be sent to script.

Format: SendChannel(channel)

Parameters:

`channel`	Can have one of following values: _CHANNEL_1 (= 1) Send events from Upstream direction of the link (channel 1) _CHANNEL_2 (= 2) Send events from Downstream direction of the link (channel 2)

Example:

```
…
SendChannel(_CHANNEL_1); # Send events from Upstream direction of the link
```

8.5 SendChannelOnly()

This function specifies that ONLY events that have occurred on the specified channel should be sent to the script.

Format: SendChannelOnly(channel)

Parameters:

`channel`	Can have one of following values: _CHANNEL_1 (= 1) Send events from Upstream direction of the link (channel 1) _CHANNEL_2 (= 2) Send events from Downstream direction of the link (channel 2)

Example:

```
…
SendChannelOnly( _CHANNEL_1 ); # Send ONLY events from Upstream
                               # direction of the link
```

8.6 DontSendChannel ()

This function specifies that events that have occurred on the specified channel should NOT be sent to the script.

Format: DontSendChannel (channel)

Parameters:

`channel`	Can have one of following values: _CHANNEL_1 (= 1) Send events from Upstream direction of the link (channel 1) _CHANNEL_2 (= 2) Send events from Downstream direction of the link (channel 2)

Example:

```
…
DontSendChannel ( _CHANNEL_1 ); # DO NOT send events from Upstream
                                # direction of the link
```

8.7 SendAllChannels()

This function specifies that events that have occurred 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 the script.

Format: SendTraceEvent(event)

Parameters:

`event` Can have one of the following values:

Packet level events:

Event value	**Description**
_PKT_DLLP	Data Link Layer Packets (DLLP)
_PKT_TLP	Transaction Layer Packets (TLP)
_PKT_ORDERED_SET	Ordered Sets
_PKT_LINK_CONDITION	Link Conditions

Link Transaction level events:

Event value	**Description**
_LINK_MEMORY	Memory transactions
_LINK_IO	IO transactions
_LINK_CONFIG	Configuration transactions
_LINK_MESSAGE	Message transactions
_LINK_COMPLETION	Completion transactions

Split Transaction level events:

Event value	**Description**
_SPLIT_MEMORY	Memory transactions
_SPLIT_IO	IO transactions
_SPLIT_CONFIG	Configuration transactions

NVM Transaction level events:

Event value	**Description**
_NVME_CONTROLLER_REG	Transactions that read or write controller register
_NVME_DOORBELL_REG	Transactions that write doorbell register.
_NVME_ADMIN_SUBMISSION_CMD	Transactions that transfer admin submission command
_NVME_NVM_SUBMISSION_CMD	Transactions that transfer NVM submission command
_NVME_COMPLETION_CMD	Transactions that transfer NVM or admin completion command
_NVME_IDX_DAT_REG	Transactions that target idx/dat registers (registers that are at fixed offset from BAR2)
_NVME_TRANSFERED_DATA	Transactions that target data which is referenced by PRP or SGL
_NVME_PRP	Generated for all PRP list transactions

_NVME_SGL	Generated for all SGL descriptors transactions

AHCI Transaction level events:

Event value	Description
_AHCI_HBA_REG	Transactions that use HBA registers
_AHCI_PORT_REG	Transactions that use Ports registers
_AHCI_CMND_LIST	Transactions that use Command list registers
_AHCI_RECEIVED_FIS	Transaction uses FIS registers
_AHCI_CMND_TABLE	Transaction uses Command table registers

Example:

```
…
SendTraceEvent( _PKT_TLP );
…

SendLevel( _LINK );
SendTraceEvent ( _LINK_MEMORY ); # Send memory Read and Write request
                                 # transactions to the script
```

8.9 DontSendTraceEvent()

This function specifies that the event specified in this function should not be sent to script.

Format: DontSendTraceEvent (event)

Parameters:

`event`	See **SendTraceEvent()** for all possible values.

Example:

```
…
SendLevel( _LINK );                  # Send Link Transaction level events
SendTraceEvent ( _LINK_CONFIG );     # Send Configuration transactions
SendTraceEvent ( _LINK_COMPLETION ); # Send Completion transactions
SendTraceEvent ( _LINK_MESSAGE );    # Send Message transactions
…
if( SomeCondition )
{
DontSendTraceEvent (_LINK_CONFIG); # Don’t send Cfg Request transactions
DontSendTraceEvent (_LINK_COMPLETION); # Don’t send Completion transactions

# Only Message transactions are sent.
}
```

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 many events are to be sent, but you need to send only one kind of event and turn off the rest.

Example:

```
…
SendLevel( _LINK );                    # Send Link Transaction level events
SendTraceEvent ( _LINK_CONFIG );       # Send Configuration  transactions
SendTraceEvent ( _LINK_COMPLETION  ); # Send Completion transactions
SendTraceEvent ( _LINK_MESSAGE  );     # Send Message transactions
…
if( SomeCondition )
{
      SendTraceEventOnly (_LINK_MEMORY );
      # Only Memory read/write request transactions 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:

```
…
SendLevel( _PACKET );    # Send packet level events
SendAllTraceEvents ( ); # All TLP, DLLP and Ordered Set packets
                        # are sent to the script
```

8.12 SendDllpType()

This function specifies more precise tuning (filtering in) for sending DLLP packets to the script.

Format: SendDllpType(dllp_type)

Parameters:

`dllp_type` Encoding of the DLLP type. This parameter may be one of the following values:

DLLP type values:

Constant	**DLLP type**
_DLLP_TYPE_ACK	Ack
_DLLP_TYPE_NAK	Nak
_DLLP_TYPE_INIT_FC1_P	InitFC1-P
_DLLP_TYPE_INIT_FC1_NP	InitFC1-NP
_DLLP_TYPE_INIT_FC1_CPL	InitFC1-Cpl
_DLLP_TYPE_INIT_FC2_P	InitFC2-P
_DLLP_TYPE_INIT_FC2_NP	InitFC2-NP
_DLLP_TYPE_INIT_FC2_CPL	InitFC2-Cpl
_DLLP_TYPE_UPDATE_FC_P	UpdateFC-P
_DLLP_TYPE_UPDATE_FC_NP	UpdateFC-NP
_DLLP_TYPE_UPDATE_FC_CPL	UpdateFC-Cpl
_DLLP_TYPE_PM	All Power Management DLLP types
_DLLP_TYPE_INVALID	Invalid DLLP types
_DLLP_TYPE_INIT_FC	All InitFC DLLP types
_DLLP_TYPE_UPDATE_FC	All UpdateFC DLLP types
_ANY_TYPE	All possible DLLP types

Example:

```
SendDllpType(_DLLP_TYPE_ACK );       # Send Ack DLLPs to the script
…

SendDllpType(_DLLP_TYPE_UPDATE_FC); # Send all UpdateFC DLLPs
```

8.13 FilterDllpType()

This function specifies more precise tuning (filtering out) for sending DLLP packets to the script.

Format: FilterDllpType(dllp_type)

Parameters:

`dllp_type`	Encoding of the DLLP type. This parameter may be one of the values defined for the **SendDllpType()** function.

Example:

```
SendDllpType(_DLLP_TYPE_INIT_FC);       # Send all InitFC DLLPs to the script
FilterDllpType(_DLLP_TYPE_INIT_FC1_CPL); # Don't send InitFCs for Completions
FilterDllpType(_DLLP_TYPE_INIT_FC2_CPL);

# Only InitFC DLLPs for Posted and Non-posted requests are sent to the script
```

8.14 SendTlpType()

This function specifies more precise tuning (filtering in) for sending TLP packets to the script.

Format: SendTlpType(tlp_type)

Parameters:

`tlp_type` Encoding of the TLP type. This parameter may be one of the following values:

TLP type values:

Constant	**TLP type**
_TLP_TYPE_INVALID	Invalid TLP types
_TLP_TYPE_MRD32	Memory Read Request, 32-bit address format
_TLP_TYPE_MRDLK32	Memory Read Request - Locked, 32-bit address format
_TLP_TYPE_MWR32	Memory Write Request, 32-bit address format
_TLP_TYPE_MRD64	Memory Read Request, 64-bit address format
_TLP_TYPE_MRDLK64	Memory Read Request – Locked, 64-bit address format
_TLP_TYPE_MWR64	Memory Write Request, 64-bit address format
_TLP_TYPE_IORD	I/O Read Request
_TLP_TYPE_IOWR	I/O Write Request
_TLP_TYPE_CFGRD_0	Configuration Read Type 0
_TLP_TYPE_CFGWR_0	Configuration Write Type 0
_TLP_TYPE_CFGRD_1	Configuration Read Type 1
_TLP_TYPE_CFGWR_1	Configuration Write Type 1
_TLP_TYPE_MSG	Message Request
_TLP_TYPE_MSGD	Message Request with Data payload
_TLP_TYPE_MSGAS	Message for Advanced Switching
_TLP_TYPE_MSGASD	Message for Advanced Switching with Data
_TLP_TYPE_CPL	Completion
_TLP_TYPE_CPLD	Completion with Data
_TLP_TYPE_CPLLK	Completion for Locked Memory Read
_TLP_TYPE_CPLDLK	Completion for Locked Memory Read with Data
_TLP_TYPE_MEMORY	All Memory Request TLP types
_TLP_TYPE_IO	All I/O Request TLP types
_TLP_TYPE_CONFIG	All Configuration Request TLP types
_TLP_TYPE_MESSAGE	All Message Request TLP types
_TLP_TYPE_COMPLETION	All Completion TLP types
_ANY_TYPE	All possible TLP types

Example:

```
SendTlpType( _TLP_TYPE_ID_MSG ); # Send Message Request TLPs to the script
…
SendTlpType( _TLP_TYPE_MEMORY ); # Send all Memory Request TLPs to the script
```

8.15 FilterTlpType()

This function specifies more precise tuning (filtering out) for sending TLP packets to the script.

Format: FilterTlpType(tlp_type)

Parameters:

`tlp_type`	Encoding of the TLP type. This parameter may be one of the values defined for the **SendTlpType()** function.

Example:

```
SendTlpType(_TLP_TYPE_CONFIG); # Send all Configuration Request TLPs
                               # to the script
FilterTlpType(_TLP_TYPE_ID_CFGRD_1); # Don't send Type 1 requests
FilterTlpType(_TLP_TYPE_ID_CFGWR_1);

# Only Type 0 Configuration Request TLPs are sent to the script
```

8.16 SendOrderedSetType()

This function specifies more precise tuning (filtering in) for sending Ordered Set packets to the script.

Format: SendOrderedSetType(set_type)

Parameters:

`set_type` Encoding of the Ordered Set type. This parameter may be one of the following values:

Ordered Set type values:

Constant	**DLLP type**
_ORDSET_TYPE_TS1	Training Sequence Type 1
_ORDSET_TYPE_TS2	Training Sequence Type 2
_ORDSET_TYPE_FTS	Fast Training Sequence
_ORDSET_TYPE_IDLE_SET	Idle Set
_ORDSET_TYPE_IDLE_GLC	Idle Glc Set
_ORDSET_TYPE_SKIP	Skip
_ORDSET_TYPE_PATN	Pattern
_ANY_TYPE	All possible Ordered Set types

Example:

```
SendOrderedSetType(_ORDSET_TYPE_FTS); # Send Fast Training Sequences
                                      # to the script
…
```

8.17 FilterOrderedSetType()

This function specifies more precise tuning (filtering out) for sending Ordered Set packets to the script.

Format: FilterOrderedSetType(set_type)

Parameters:

`set_type`	Encoding of the Ordered Set type. This parameter may be one of the values defined for the **SendOrderedSetType()** function.

Example:

```
FilterOrderedSetType(_ORDSET_TYPE_SKIP); # Don’t send Skip packets.
```

9 Timer Functions

This group of functions covers VSE capability to work with timers –-- internal routines that repeatedly measure a timing interval between different events.

9.1 VSE Time Object

A VSE time object is a special object that presents time intervals in verification scripts. From point of view of the **CSL**, the verification script time object is a “list”-object of two elements. (Please see the *CSL Manual* for more details about CSL types.)

[seconds, nanoseconds]

Note: The best way to construct a VSE time object is to use the **Time()** function (see below).

9.2 SetTimer()

Starts the timing calculation from the event where 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 calculations again from the point where it was called.

9.3 KillTimer()

Stops 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.
```

9.4 GetTimerTime()

Retrieve the timing interval from the specific timer.

Format: GetTimerTime (timer_id = 0)

Parameters:

`timer_id`	Unique timer identifier

Return values:

Returns VSE time object from 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.

10 Time Construction Functions

This group of functions are used to construct VSE time objects.

10.1 Time()

Constructs a verification script time object.

Format: Time(nanoseconds)
Time(seconds, nanoseconds)

Return values:

First function returns **[0, nanoseconds]**
Second function returns **[seconds, nanoseconds]**

Parameters:

`nanoseconds`	Number of nanoseconds in specified time
`seconds`	Number of seconds in specified time

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.**

11 Time Calculation Functions

This group of functions covers VSE capability to work with "time" – VSE time objects.

11.1 AddTime()

Adds two VSE time objects

Format: AddTime(time1, time2)

Return values:

Returns VSE time object representing the time interval equal to the sum of **time_1** and **time_2.**

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t2 = Time(2, 200);
t3 = AddTime( t1, t2 ) # Returns VSE time object = 2 sec 300 ns.
```

11.2 SubtractTime()

Subtract two VSE time objects

Format: SubtractTime (time1, time2)

Return values:

Returns VSE time object representing the time interval equal to the difference between **time_1** and **time_2.**

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t2 = Time(2, 200);
t3 = SubtractTime ( t2, t1 ) # Returns VSE time object = 2 sec 100 ns.
```

11.3 MulTimeByInt()

Multiplies VSE time object by integer value

Format: MulTimeByInt (time, mult)

Return values:

Returns VSE time object representing the time interval equal to the product of **time * mult.**

Parameters:

`time`	VSE time object
`mult`	multiplier, integer value

Example:

```
t  = Time(2, 200);
t1 = MulTimeByInt ( t, 2 ) # Returns VSE time object = 4 sec 400 ns.
```

11.4 DivTimeByInt()

Divides VSE time object by integer value

Format: DivTimeByInt (time, div)

Return values:

Returns VSE time object representing the time interval equal to the quotient of **time / div.**

Parameters:

`time`	VSE time object
`div`	divisor, integer value

Example:

```
t  = Time(2, 200);
t1 = DivTimeByInt ( t, 2 ) # Returns VSE time object = 1 sec 100 ns.
```

12 Time Logical Functions

This group of functions covers VSE capability to compare VSE time objects

12.1 IsEqualTime()

Verifies that one VSE time object is equal to the other VSE time object.

Format: IsEqualTime (time1, time2)

Return values:

Returns 1 if **time_1** is equal to **time_2**, returns 0 otherwise.

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t2 = Time(500);
If( IsEqualTime( t1, t2 ) ) DoSomething();
```

12.2 IsLessTime()

Verifies that one VSE time object is less than the other VSE time object

Format: IsLessTime (time1, time2)

Return values:

Returns 1 if **time_1** is less than **time_2**, returns 0 otherwise.

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t2 = Time(500);
If( IsLessTime ( t1, t2 ) ) DoSomething();
```

12.3 IsGreaterTime()

Verifies that one VSE time object is greater than the other VSE time object

Format: IsGreaterTime (time1, time2)

Return values:

Returns 1 if **time_1** is greater than **time_2**, returns 0 otherwise.

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t2 = Time(500);
If( IsGreaterTime ( t1, t2 ) ) DoSomething();
```

12.4 IsTimeInInterval()

Verifies that a VSE time object is greater than some VSE time object and less than the other VSE time object.

Format: IsTimeInInterval(min_time, time, max_time)

Return values:

Returns 1 if **min_time <= time <= max_time**, returns 0 otherwise.

Parameters:

`time_1`	VSE time object representing the first time interval
`time_2`	VSE time object representing the second time interval

Example:

```
t1 = Time(100);
t  = Time(400);
t2 = Time(500);
If( IsTimeInInterval ( t1, t, t2 ) ) DoSomething();
```

13 Time Text Functions

This group of functions covers VSE capability to convert VSE time objects into text strings.

13.1 TimeToText()

Converts a VSE time object into text.

Format: TimeToText (time)

Return values:

Returns a text representation of VSE time object

Parameters:

`time`	VSE time object

Example:

```
t = Time(100);
ReportText( TimeToText( t ) ); # See below details for ReportText() function
```

14 Output Functions

This group of functions covers VSE capability to present information in the output window.

14.1 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_frames = in.NumOfFrames;
text = Format( “Number of frames : %d”, num_of_frames );
ReportText ( text );
…
x = 0xAAAA;
y = 0xBBBB;
text = FormatEx( “x = 0x%04X, y = 0x%04X”, x, y );
ReportText( “Text : ” + text );
…
```

14.2 EnableOutput()

Enables showing information in the output window and sending COM reporting notifications to COM clients.

Format: EnableOutput ()

Example:

```
EnableOutput ( );
```

14.3 DisableOutput()

Disables showing information in the output window and sending COM reporting notifications to COM clients.

Format: DisableOutput ()

Example:

```
    DisableOutput ();
```

15 Information Functions

15.1 GetTraceName()

This function returns the filename of the trace file being processed by VSE.
If the script is being run over a multi-segmented trace, this function returns the path to the segment being processed.

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 name : ” + GetTraceName() );
  …
  File = OpenFile( "C:\\My Files\\" + GetTraceName(1) + "_log.log" );

# For trace file with path - D:\Some Traces\Data.pex
# GetTraceName(1) returns – "D_Some Traces_Data.pex"
```

15.2 GetScriptName()

This function returns the name of the verification script where this function is called.

Format: GetScriptName()

Example:

```
ReportText( “Current script : ” + GetScriptName() );
```

15.3 GetApplicationFolder()

This function returns the full path of the folder where the PCIe Protocol Suite™ application was started.

Format: GetApplicationFolder()

Example:

```
ReportText( "PCIe Protocol Suite folder : " + GetApplicationFolder () );
```

15.4 GetCurrentTime()

This function returns the string representation of the current system time.

Format: GetCurrentTime()

Example:

```
ReportText( GetCurrentTime() ); # Yields “February 10, 2004, 5:49 PM”
```

15.5 GetEventSegNumber()

In case if a multi-segmented trace is being processed, this function returns the index of the segment for the current event.

Note: When a multi-segmented trace file (extension ***.pem**) is processed by VSE, different trace events in different segments of the same trace file may have the same indexes (value stored in **in.Index** input context members), but they have different segment numbers.

Format: GetEventSegNumber()

Example:

```
ReportText( Format( “Current segment = %d”, GetEventSegNumber() ) );
```

15.6 GetTriggerPacketNumber()

This function returns the number of the trigger packet in the trace. In case no trigger event was recorded in the trace, a value of 0xFFFFFFFF is returned.

Format: GetTriggerPacketNumber()

Example:

```
ReportText( FormatEx( “Trigger packet # : %i”, GetTriggerPacketNumber() );
```

15.7 TraHasError ()

This function returns non-zero value if transaction has passed error.

Format: TraHasError(error_code)

Example:

```
if ( TraHasError( _NVMC_ERROR_INCOMPLETE_SUB_TRA ) )
{
        ReportText("Incompete sub transaction") );
}
```

16 Navigation Functions

16.1 GotoEvent()

This function forces the application to jump to some trace event and show it in the main trace view.

Format: GotoEvent(level, index, segment)
GotoEvent()

Parameters:

`level`	Transaction level of the event to jump to (possible values: _PACKET, _LINK, _SPLIT, _NVME, _NVMC, _AHCI, _ATA)
`index`	Transaction index of the event to jump to
`segment`	Segment index of the event to jump to. If omitted, the current segment index is used.

Remarks:

If no parameters were specified, the application jumps to the current event being processed by VSE. The **segment** parameter is used only when the verification script is running over a multi-segmented trace (extension: ***.pem**). For regular traces it is ignored.

If wrong parameters were specified (like an index exceeding the maximum 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_segment = GetEventSegNumber();
  interesting_level = in.Level;
  interesting_index = in.Index;
 }
 …
OnFinishScript()
{
     …
      # go to the interesting event…
      GotoEvent( interesting_level, interesting_index, interesting_segment );
}
```

16.2 SetMarker()

This function sets a marker for some trace event.

Format: **SetMarker(marker_text)**
SetMarker(marker_text, level, index, segment)

Parameters:

`marker_text`	Text of the marker
`level`	Transaction level of the event to jump to (possible values: _PACKET, _LINK, _SPLIT, _NVME, _NVMC, _AHCI, _ATA)
`index`	Transaction index of the event to jump to
`segment`	Segment index of the event to jump to. If omitted, the current segment index is used.

Remarks:

If no parameters were specified, other than **marker_text,** the application sets a marker to the current event being processed by VSE. The **segment** parameter is used only when a verification script is running over a multi-segmented trace (extension: ***.pem**). For regular traces it is ignored.

If wrong parameters were specified (like an index exceeding the maximum index for a 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_segment = GetEventSegNumber();
 interesting_level = in.Level;
 interesting_index = in.Index;
}
…
OnFinishScript()
{
    …
     # set marker to the interesting event…
     SetMarker(  "  !!! Cool Marker !!! ", interesting_level,
                                           interesting_index,
                                           interesting_segment );

     # go to the interesting event…
     GotoEvent( interesting_level, interesting_index, interesting_segment );
}
```

17 File Functions

This group of functions covers VSE capabilities to work with the external files.

17.1 OpenFile()

This function opens a file for writing.

Format: OpenFile(file_path, append)

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 to the contents of the file. Otherwise, the contents of the file are overwritten.

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, 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
…
# Opens file, the following file operations append to contents of the file.
file_handle = OpenFile( GetApplicationFolder() + "Log.txt", _APPEND );
```

17.2 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
…
```

17.3 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”

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
…
```

17.4 ShowInBrowser()

This function allows you to open 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 extensions ***.htm** and the file handle passed to **ShowInBrowser()** function belongs to a file with such an extension,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 );
…
```

18 COM/Automation Communication Functions

This group of functions covers VSE capabilities to communicate with COM/Automation clients connected to the PCIe Protocol Suite™ application. (Please refer to the *PCIe Protocol Suite Automation Manual* for the details on how to connect to the PCIe Protocol Suite application and VSE)

18.1 NotifyClient()

This function allows you to send information to COM/Automation client applications in a custom format. The client application receives a VARIANT object, which it is supposed to parse.

Format: NotifyClient(param_list)

Parameters:

`param_list`	List of parameters to be sent to the client application. Each parameter might be an integer, string or list. (See *CSL Manual* for details about data types available in CSL.) Because the list itself may contain integers, strings, or other lists – it is possible to send complicated messages. (Lists should be treated as arrays of VARIANTs.)

Example:

```
…
if( SomeCondition() )
{
      NotifyClient( 2, [ in.Index, in.Level, “CHANNEL 2”, “TLP”,
                                   TimeToText( in.Time )] );
}
…
# Here we sent 2 parameters to clients applications :
#  2 ( integer ),
# [ in.Index, in.Level, “CHANNEL 2”, “TLP”, TimeToText( in.Time )] ( list )
```

Remark:

See an example of handling this notification by client applications and parsing code in the *PE Automation* document.

19 User Input Functions

19.1 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 you omit the title, 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 the user 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 the GUI. For VSEs controlled by COM/Automation clients, it does nothing.

This function "locks" the PCIe Protocol Suite™ application, which means that there is no access to other application features until the dialog box is closed. In order to prevent too many **MsgBox** calls -- in the case of a script not written correctly – VSE keeps track of all function calls demanding user interaction and doesn't show dialog boxes if a customizable limit was 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…
    }
```

19.2 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 (see the **CSL** manual for details about list objects) or a 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 you omit **title**, 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 you omit **default_text**, 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 upon the **return_type** argument, this function returns either a CSL list object or the text typed in the text box as it is.

In case of **return_type = _IB_LIST** (by default), the text in the text box is considered as a set of list items delimited by ',' (only hexadecimal, decimal, and string items are currently supported).

Text example:

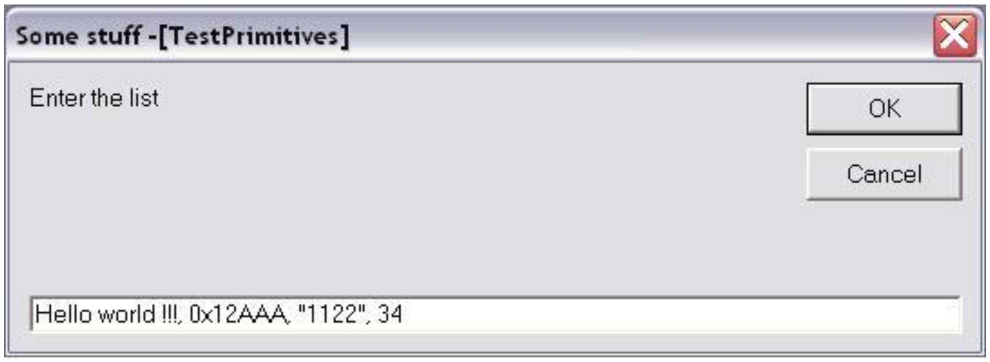
```
Hello world !!!, 12, Something, 0xAA, 10, "1221"
```

Produces a CSL list object of5 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, a text enclosed in quote signs "" is always considered as a string.



Hello world !!!, 0x12AAA, "1122", 34

Remark:

This function works only for VS Engines controlled via the GUI. For VSEs controlled by COM/Automation clients, it does nothing.

This function "locks" the PCIe Protocol Suite application, which means that there is no access to other application features until the dialog box is closed. In order to prevent too many **InputBox** calls -- in the case of a script not written correctly – VSE keeps track of all function calls demanding user interaction and doesn't show dialog boxes if a customizable limit was exceeded (returns **null** object in that case).

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 );
    }
```

19.3 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
 …
```

19.4 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 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
 …
```

20 String Manipulation/Formating Functions

20.1 FormatEx()

Write formatted data to a string. **FormatEx()** 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

Remark:

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 0**X** 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"
```

21 Miscellaneous Functions

21.1 ScriptForDisplayOnly()

Specifies that the script is designed for displaying information only and that its author doesn't care about verification script result. Such a script has a result of **DONE** after execution.

Format: ScriptForDisplayOnly ()

Example:

```
ScriptForDisplayOnly();
```

21.2 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 sleeping time.

Format: Sleep(time)

Parameters:

`time`	VSE time object specifying sleep time

Example:

```
Sleep ( Time(1000) );
# Don't send any event occurred during 1 ms from the current event.
```

21.3 ConvertToHTML()

This function replaces spaces with "&nbsp" and carriage return symbols with "<br>" 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&nbsp!!!<br>How&nbspare&nbspyou&nbsptoday?"
```

Note : Some other useful miscellaneous functions can be found in the file **VSTools.inc.**

21.4 Pause()

Pauses a running script. Later, script execution can be resumed or cancelled.

Format: Pause()

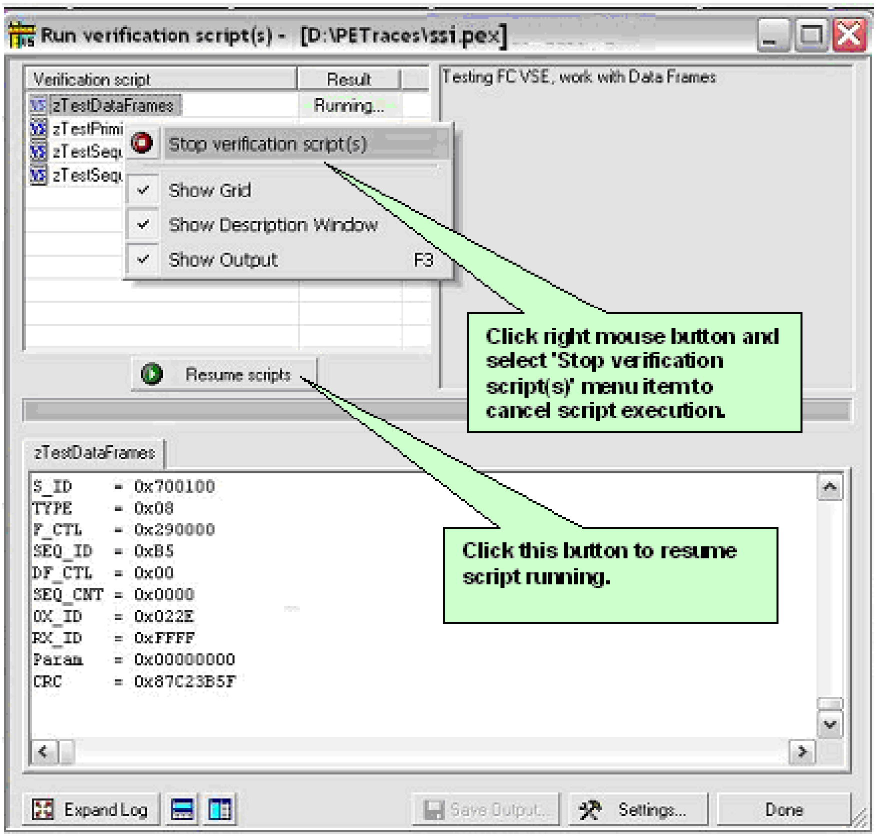
Example:

```
…
If( Something_Interesting() )
{
      GotoEvent(); # Jump to the trace view
      Pause();     # Pause script execution
}
…
```

Remark:

This function works only for 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:


Run verification script(s) - [D:\PETraces\ssi.pex]
Verification script
Result
zTestDataFrames
Running...
zTestPrimi
zTestSequ
zTestSequ
Testing FC VSE, work with Data Frames
Stop verification script(s)
Show Grid
Show Description Window
Show Output F3
Click right mouse button and select 'Stop verification script(s)' menu item to cancel script execution.
Resume scripts
Click this button to resume script running.
zTestDataFrames
Expand Log
Save Output...
Settings...
Done

```
S_ID    = 0x700100
TYPE    = 0x08
F_CTL   = 0x290000
SEQ_ID  = 0xB5
DF_CTL  = 0x00
SEQ_CNT = 0x0000
OX_ID   = 0x022E
RX_ID   = 0xFFFF
Param   = 0x00000000
CRC     = 0x87C23B5F
```

22 The VSE Important Script Files

The VSE working files are located in the **..\Scripts\VFScripts** subfolder of the main PCIe Protocol Suite™ folder. The current version of VSE includes the following files:

File	Description
VSTools.inc	Main VSE file containing definitions of some generic and PCI Express-specific VSE script functions provided by Teledyne LeCroy (must be included in every script). **NOTE**: The files VS_constants.inc and VS_Primitives.inc are included.
VS_constants.inc	File containing definitions of some important generic and PCI Express-specific VSE global constants
VSTemplate.pev_	Template file for new verification scripts.
VSUser_globals.inc	File of user global variable and constant definitions (In this file, it is useful to enter definitions of constants, variables, and functions to be used in many scripts you write.)

22.1 Example Script Files

The VSE example files are located in the **..\Scripts\VFScripts\Samples** subfolder of the main PCIe Protocol Suite folder. The current version of VSE includes the following files:

File	Description
examp_tlp_data.inc	Sample include file containing definitions and functions used by some other sample scripts
examp_dllps.pevs	Sample processing script that outputs information about DLLP packets present in the trace
examp_tlps.pevs	Sample processing script that outputs information about TLP packets present in the trace
examp_ordered_sets.pevs	Sample processing script that outputs information about Ordered Set and Link Condition packets present in the trace
examp_check_errors.pevs	Sample PASS/FAIL script that checks all packets in the trace for all the errors VSE exports and fails in case any error is found
examp_link_transactions.pevs	Sample processing script that outputs information about Link Transactions present in the trace
examp_split_transactions.pevs	Sample processing script that outputs information about Split Transactions present in the trace
examp_metrics.pevs	Sample processing script that outputs information about Memory Write Link Transaction metrics and all Split Transaction metrics
examp_nvme.pevs	Sample processing script that outputs information about NVM Transactions present in the trace
examp_nvme_errors.pevs	Sample processing script that outputs information about NVM Transaction errors present in the trace
examp_nvmc.pevs	Sample processing script that outputs information about NVM Commands present in the trace
examp_nvmc_errors.pevs	Sample processing script that outputs information about NVM Commands errors present in the trace
examp_nvc_deltatime_metrics.pevs	Sample processing script that outputs information about NVM Commands delta time metrics present in the trace
examp_ahci.pevs	Sample processing script that outputs information about AHCI Transactions present in the trace
examp_ata.pevs	Sample processing script that outputs information about ATA Transactions present in the trace
examp_ahci_errors.pevs	Sample processing script that outputs information about AHCI Transaction errors present in the trace
examp_ata_errors.pevs	Sample processing script that outputs information about ATA Transaction errors present in the trace

How to Contact Teledyne LeCroy

Type of Service	Contact
Call for technical support…	US and Canada: 1 (800) 909-7112 Worldwide: 1 (408) 653-1260
Fax your questions…	Worldwide: 1 (408) 727-6622
Write a letter …	Teledyne LeCroy Corporation Protocol Solutions Group Customer Support 3385 Scott Blvd. Santa Clara, CA 95054-3115
Send e-mail…	psgsupport@teledynelecroy.com
Visit Teledyne LeCroy web site…	teledynelecroy.com/