NET PROTOCOL SUITE

Verification Scripting Engine (VSE) API

Reference Manual

For Net Protocol Suite software version 6.65

Generated: February 27, 2026, 13:55

Teledyne LeCroy Protocol Solutions Group

Trademarks and Servicemarks

Teledyne LeCroy, CATC Trace, SierraNet T328, SierraNet M328, SierraNet M408, SierraNet M168, SierraFC M164, SierraFC M8-4 and NetProtocolSuite are trademarks of Teledyne LeCroy.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

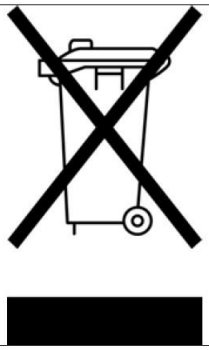
Intel and Pentium are registered trademarks of Intel Corporation.

All other trademarks and registered trademarks are property of their respective owners.

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE
WITHOUT NOTICE. ALL INFORMATION, EXAMPLES AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE REPRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS ARE FULLY RESPONSIBLE FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN INFORMATION THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT Teledyne LeCroy FOR A COPY.

WEEE Program



This electronic product is subject to disposal and recycling regulations that vary by country and region. Many countries prohibit the disposal of waste electronic equipment in standard waste receptacles. For more information about proper disposal and recycling of your Teledyne LeCroy product, please visit teledynelecroy.com/recycle.

Teledyne LeCroy
teledynelecroy.com

Contents

Chapter 1

Verification Scripting Engine (VSE) API

This is a reference manual for the Verification Scripting Engine (VSE) Application Programming Interface (API).

1.1 Introduction

VSE allows users to ask the application to send some desired "items" (any frames, packets, ordered sets, commands, etc.) from a trace file to a verification script written using the VSE script API. This script then evaluates the sequence of items (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.

The VSE API was designed to allow users to easily retrieve information about any field in a packet/frame/command, and to make complex timing calculations between different items in a saved trace. It also allows filtering-in or filtering-out of data with dynamically changing filtering conditions, outputting of information to a special output window, and setting bookmarks in the trace.

1.2 VSE Script Structure Overview

1.2.1 Python Usage

The base script language of VSE is Python 3.5, so you may import any compatible Python libraries into your VSE script in order to implement custom integrations with your other tools and systems. However, a VSE script cannot be executed in a stand-alone Python console; it can only be executed from within the Net Protocol Suite application or the Python API. Also a standalone Python and pip are installed along application to support third party Python library by VSE. You can find the Python folder in “C:\Users\Public\Documents\LeCroy\Net Protocol Suite\Python”. So, you must use the shipped python to install/update any python library. For example, to install “nose” one should follow the following on a command line:

Windows:

```
C:\
Cd C:\Users\Public\Documents\Lecroy\Net Protocol Suite\Python
Python.exe –m pip install nose.
```

Linux:

```
pip3 install --target=/usr/local/LeCroy/Shared.General/python/python35-
x86_64/lib/python3.5    module_name
```

For example, if you want to install “nose”, you need to replace module_name with nose.

Note: There is a Python known issue in “numpy” that will cause the application to crash at exit if “numpy” is imported directly or indirectly to the VSE script. It has been reported to the Python which you can read more in https://github.com/numpy/numpy/issues/8097

Note: Importing TLNetAPI.lib in VSE script is not allowed. It means user cannot use API functions inside VSE script.

1.2.2 Global Objects

A script has access to the following pre-defined global objects:

1. Trace
   It has various properties and functions related to the entire trace being processed.
2. Output
   It enables your script to send feedback/results back to the application.

See the full definitions in "Global Objects" on page 8 below.

1.2.3 Required Elements

There are a few elements that you must add to any VSE Script.

1. VSE Configuration
   You must create a configuration object (CVSEConfig) and set various parameters that control how a trace will be processed. See the full definitions in "Classes" on page 9 below.
2. "OnItem" Callback
   You must define a callback function for handling items. When called, it is passed a CTraceItem object that you will use to extract details about each item. See the full definition in the Trace.ProcessVSE description in Table 1.14 on page 8 below.
3. Trace.ProcessVSE(…)
   This is the main function that starts the processing. You must call this function on the global Trace object and pass it your configuration object and your "OnItem" callback. It will handle traversal through the trace according to your configuration settings, and it will call your "OnItem" callback function for each item that matches your configuration settings. See the full definition in the Trace.ProcessVSE description in Table 1.14 on page 8 below.
4. Output.Result
   Set the Result property on the Output global object at the end of your script. This feedback will be sent back to the application so that it can show the script results. See the full definition of Output in "Output" on page 8 below.

Note: It is supported to call Trace.ProcessVSE multiple times in a single script (possibly with a different configuration and callback each time), thereby enabling you to create complex, multi-pass processing scripts.

1.3 API Definitions: Objects and Types

The verification scripting API is made of a list of VSE objects and types as defined below:

1.3.1 Enumerations

TABLE 1.1: Enum VSEResult

Type	Name	Result	Comments
Enum Item	NoResult	NA	Result is not set.
Enum Item	Success	NA	Successful
Enum Item	Failure	NA	Failed

TABLE 1.2: Enum VSEPort

Type	Name	Result	Comments
Enum Item	AllExcept (Default)	NA	VSE will traverse all ports except those which are removed using DontSendPort ()
Enum Item	NoneExcept	NA	VSE won't traverse any port except those which are added using SendPort ()

Note: To identify a single port, use positive integers starting from 1. E.g., 1 corresponds to "P1", 2 corresponds to "P2", etc.

TABLE 1.3: Enum VSEItemType:

Type	Name	Result	Comments
Enum Item	AllExcept (Default)	NA	VSE will traverse all events except those which are removed using DontSendItem () or DontSendPort or DontSendPort_2
Enum Item	NoneExcept	NA	VSE won't traverse any event except those which are added using SendItem () or SendPort or SendPort_2
Enum Item	...	NA	See GEVSEItemType for specific items for this enumerator.

TABLE 1.4: Enum TraceTraverseOrder:

Type	Name	Result	Comments
Enum Item	Linear (Default)	NA	Traverse events in order of event's timestamp.
Enum Item	Transaction	NA	Traverse based on actual transaction order. (Command, data, response)

TABLE 1.5: Enum TraceTraverseType:

Type	Name	Result	Comments
Enum Item	Event (Default)	NA	Item in each iteration consists of all packets in the event (frame).
Enum Item	Packet	NA	Item in each iteration only have one packet at a time. So for example, for a SCSI command event, 3 items will be sent. One for each of command, transport and link.

TABLE 1.6: Enum ByteOrder:

Type	Name	Result	Comments
Enum Item	Unknown	NA	Item's endianness is not defined.
Enum Item	LittleEndian	NA	Little endian item
Enum Item	BigEndian	NA	Big (Network) endian item.

TABLE 1.7: Enum BitOrder:

Type	Name	Result	Comments
Enum Item	LSBFirst	NA	Least significant bit appears first in the item's data.
Enum Item	MSBFirst	NA	Most significant bit appears first in the item's data.

TABLE 1.8: Enum GEVSEItemType: of type VSEItemType

Type	Name	Result	Comments
Enum Item	FC	NA	An FC frame.
Enum Item	Ethernet	NA	An Ethernet frame.
Enum Item	Physical	NA	A link event, such as "Loss of Sync" or Auto-negotiation or Link training.
Enum Item	Primitive	NA	A primitive or ordered set.

TABLE 1.9: Enum GETraceItemError:

Type	Name	Result	Comments
Enum Item	DELIMITER_ERROR	NA	The frame ended without any termination block, or EOF on the line without any received SOF, or SOF on the line if the following symbol is a control symbol.
Enum Item	FRAME_LENGTH_ERROR	NA	Frames with under/over length cause this error. A frame length is the number of bytes between SOF and EOF. When this value is less than 28 bytes (24 bytes for frame Header length and 4 bytes for CRC), it is considered as under-length error. For over-length error, this value shall not exceed 2112 Bytes, excluding Optional-Extended-Header, 24 bytes of frame Header length and 4 bytes of CRC.
Enum Item	ETHERNET_CRC_ERROR	NA	Ethernet FCS Error
Enum Item	FC_CRC_ERROR	NA	One or more CRC errors occurred with the Link Layer since the bit was last cleared. If the Transport receives an Frame with an invalid CRC signaled from the Link layer, the Transport layer shall signal the Link layer to negatively acknowledge frame reception by asserting error during the frame acknowledgment frame reception by asserting error during the frame acknowledgment handshake.
Enum Item	BLOCK_TYPE_ERROR	NA	Block Type Error
Enum Item	ORDER_SET_ERROR	NA	Ordered Set Error
Enum Item	ALIGNMENT_ERROR	NA	Align-primitives on all the 4 lanes do not occur at the same time.
Enum Item	SYNC_HEADER_ERROR	NA	If both bits in the Synchronization Header have the same value, the Transmission Word shall cause a code violation to be reported and shall be decoded as two Idle Special Functions.
Enum Item	FEC_ERROR	NA	Uncorrectable Forward Error Correction Error
Enum Item	AUTO_NEG_FRAME_MARKER_ERROR	NA	Frame Marker Error for Auto Negotiation frames

Type	Name	Result	Comments
Enum Item	AUTO_NEG_MANCHESTER_ERROR	NA	Manchester Violation Error for Auto Negotiation frames
Enum Item	MARKER_INTERVAL_ERROR	NA	The timing between 2 align-primitives on the same lane is not acceptable.
Enum Item	TRAINING_FRAME_MARKER_ERROR	NA	Frame Marker Error for Training Sequence frames
Enum Item	TRAINING_MANCHESTER_ERROR	NA	Manchester Violation Error for Training Sequence frames
Enum Item	TCP_UDP_CHECKSUM_ERROR	NA	There is a check sum error in TCP or UDP frame
Enum Item	CHECKSUM_ERROR	NA	This error set by software if during capturing trace, user sets "Detect Checksum error" in preferences.It checks checksum for TCP/UDP/ICMP/IGMP/IP/MPA/OSPF
Enum Item	ECN_ERROR	NA	There is an ECN error.
Enum Item	IPG_ERROR	NA	There is an IPG error.

TABLE 1.10: Enum FCTraceItemError:

Type	Name	Result	Comments
Enum Item	SYMBOL_VIOLATION	NA	For FC 8G traffic: Invalid 10-bit symbol. Detection of a code violation does not necessarily indicate that the transmission character in which the code violation was detected is in error. Code violations may result from a prior error that altered the running disparity of the bit stream but did not result in a detectable error at the transmission character in which the error occurred. For FC 16 and 32G traffic: 64/66b block type error.
Enum Item	DISPARITY_ERROR	NA	Incorrect disparity was detected one or more times since the last time the bit was cleared.
Enum Item	SPACING_ERROR	NA	A spacing error in which fewer than two Fill Word (Idle/ARBFF or any Unknown Primitives) are received between frames. These primitives are necessary to allow for maintaining link

Type	Name	Result	Comments
			synchronization in the absence of other transmission words.
Enum Item	ALIGNMENT_ERROR	NA	An error in the 32-bit data alignment.
Enum Item	DELIMITER_ERROR	NA	A sequence error in the frame delimiters SOF and EOF. If two SOF delimiters or two EOF delimiters are received in a row, this error will be flagged.
Enum Item	EOF_ERROR	NA	Receiving an invalid EOF at end of frame that may include EOFa or EOFni cause an error in EOF.
Enum Item	PRIMITIVE_ERROR	NA	This is caused by receiving an undefined primitive.
Enum Item	FRAME_LENGTH_ERROR	NA	Frames with under & over length cause error as frame length, by counting frame length between SOF and EOF. For under length it should not be less than 24+4: (24 bytes for Header Length and 4 bytes for CRC. For Over Length total length between SOF and EOF minus Optimal-ExtendedHeader (if exist), minus Frame-Header-Length (24 Bytes), minus CRC-Length (4 Byte) should not be exceed more than 2112 Bytes.
Enum Item	CRC_ERROR	NA	One or more CRC errors occurred with the Link Layer since the bit was last cleared. If the Transport receives an Frame with an invalid CRC signaled from the Link layer, the Transport layer shall signal the Link layer to negatively acknowledge frame reception by asserting error during the frame acknowledgment handshake.
Enum Item	SYNC_HEADER_ERROR	NA	If both bits in the Synchronization Header have the same value, the Transmission Word shall cause a code violation to be reported and shall be decoded as two Idle Special Functions.
Enum Item	FEC_PARITY_ERROR	NA	FEC Parity Error
Enum Item	ILLEGAL_PRIMITIVE_ERROR	NA	On 16G traffic ARB (ARBff, ARBval , and ARByx) primitives are illegal and vendor specific BB_SCs and BB_SCr primitives are illegal on all speeds.

Type	Name	Result	Comments
Enum Item	TRAINING_FRAME_MARKER_ERROR	NA	Frame Marker Error for Training Sequence frames
Enum Item	TRAINING_MANCHESTER_ERROR	NA	Manchester Violation Error for Training Sequence frames

TABLE 1.11: Enum GELogicalField:

Type	Name	Result	Comments
Enum Item	START_TIME	Integer 64	The start timestamp of the item in nanoseconds.
Enum Item	DURATION_TIME	Integer 64	The time duration of the item in nanoseconds.
Enum Item	ANALYZER_SPEED	String	The bit rate of the item. (1G, 2G, 4G, 8G, 10G, 16G, 25G, 32G, 40G, 50G, 100G)
Enum Item	PORT_NO	String	The alias of the port on which the item was captured. By default, "P1", "P2", etc.
Enum Item	PORT_NO_2	String	This logical field returns port alias name string according to exact port label on the board. It means for QSFP ports, it returns “P9”, “P10”. Actually, string value of this field is same as “Port No” column in viewer.
Enum Item	CURRENT_STATE	String	The analyzer sequencer state number at the time the item was captured.
Enum Item	ETHERNET_TAG	String	For Ethernet frame items, the VLAN tag name(s), if any. If a frame has multiple tags, they will be concatenated together like "Tag1 - Tag2 - Tag3 ....". Possible tag names are: "ISL", "VXLAN", "GRE", "NVGRE", or any 802.3 tag name.
Enum Item	DATA_LENGTH	Integer 32	For frame items, the length (in bytes) of the data payload.
Enum Item	PROTOCOL_TYPE	String	"Ethernet" or "FC".
Enum Item	PACKET_LENGTH	Integer 32	The length in bytes of the entire item. The value is formatted as a string like "<length> Bytes".
Enum Item	LANE_NO	Integer 32	For link training frames at 40/50/100 G, the physical lane on which it was captured.

Type	Name	Result	Comments
Enum Item	FEC_STATUS	String	The FEC Mode of the link. ("", "BASE-R FEC", or "Reed-Solomon FEC")
Enum Item	COUNT	Integer 32	For items with a repeat count (AN, Training, Ordered Sets), the number of instances captured.
Enum Item	JAMMER_PORT_NAME	String	For traces from a jammer configuration, an indication of whether item was captured before/ after jammer. ("After Jam" or "Before Jam")
Enum Item	LINK_FUNCTION_NAME	String	A decoded abstract of the item, depending on its type. It corresponds to the value shown in the Frame column of the application's Spreadsheet View.
Enum Item	TRANSPORT_FUNCTION_NAME	String	A decoded abstract of the item, depending on its type. It corresponds to the value shown in the Transport Function Name column of the application's Spreadsheet View.
Enum Item	RESPONSE_TIME	Integer 32	Response time of an exchange in nanosecond.This value is valid for first frame of an exchange, for other frame of exchange it is not applicable.
Enum Item	LATENCY	Integer 32	Latency time of an exchange in nanosecond. This value is valid for first frame of an exchange, for other frame of exchange it is not applicable.
Enum Item	PENDING_EXCHANGES	Integer 32	Number of pending exchanges. This value is valid for first and last frame of exchange, for other frames of exchange it is not applicable.
Enum Item	EXCHANGE_STATUS	String	Status of an exchange as below: “Success;” “Fail;” “Incomplete;”
Enum	IPG_DURATION	Integer 32	Interpacket Gap time in nanosecond.

TABLE 1.12: Enum ExchangeState:

Type	Name	Result	Comments
Enum Item	Start	None	If item is first frame of exchange, this value returned by ExchangeStatus property
Enum Item	End	None	If item is last frame of exchange, this value returned by ExchangeStatus property

Type	Name	Result	Comments
Enum Item	`Mid`	`None`	If item is middle frame of exchange, this value returned by ExchangeStatus property
Enum Item	`NA`	`None`	If traverse order is linear, NA returned by ExchangeStatus for all items.

1.3.2 Global Objects

TABLE 1.13: Output

Type	Name	Result	Comments
R/W Property	`Result`	`VSEResult`	Set by user to indicate the result of VSE.
Function	`SetProgress(min, max, pos)`	`None`	Set minimum, maximum and current position of progress.
Function	`AppendLog(message)`	`None`	Append message to the VSE log control.
Function	`AppendHTMLLog(html_message)`	`None`	Append HTML message to the VSE log control. **Note1**: This function won't work if AppendLog() is called in the same script. **Note2**: AppendHTMLLog does not support complex rich text rendering with tables and floats, it does support limited paragraph-based formatting that you may need in a log viewer.
Function	`ClearLog()`	`None`	Clear the log if applicable. Has no effect on log file.

TABLE 1.14: Trace

Type	Name	Result	Comments
R/O Property	`Name`	`String`	Trace file path name.
R/O Property	`ItemCount`	`Integer`	Number of items (Links) within trace.
R/O Property	`TriggerIndex`	`Integer`	Trace's trigger index (Link index). The returned index is 1-based.

Type	Name	Result	Comments
R/O Property	`TriggerTimeStamp`	`Double`	Double precision floating point seconds since start to trace's trigger.
R/O Property	`PortList`	`List`	Returns list of existing ports in the trace. Here is port mapping: P1: 1 P2: 2 …… P8 : 8 P9 : 1 P10: 2 When there are more than one board, port index increased sequentially for second board.
R/O Property	`PortList_2`	`List of List`	Returns list of pairs. Each pair specifies board index (one base) and port label on the board.
Function	`ProcessVSE(config, on_item)`	`None`	This function will call on_item function for each item in the trace according to the config parameter (See CVSEConfig). It will continue until there is no item remaining or on_item returns False. on_item is a function with signature Bool(CVSEItem).
Function	`GetItemByIndex(event_number)`	`Item`	Retrieves an Item specified by event_number (Link index). event_number is 1-based.
Function	`GetItemByPacketKey(packet_key)`	`Item`	Retrieves an Item specified by packet_key.
Function	`SetBookmark(packet_key, name, desc)`	`None`	Creates a bookmark at specified packet key.
Function	`RemoveAllBookmarks`	`None`	Remove all bookmarks from trace.
Function	`RemoveBookmark (packet_key)`	`Boolean`	Remove bookmark from given packet_key.

Type	Name	Result	Comments
Function	GetAllBookmarks	CTraceBookmark[]	Retrieves list of all bookmarks in trace. See Table 1.19

1.3.3 Classes

TABLE 1.15: Class CPacketKey:

Type	Name	Result	Comments
R/O Property	IsValid	Boolean	Indicates whether the packet key is valid or not
R/W Property	Layer	Integer	Layer specifies layer ID of packet. You can see layer ID enumeration in TABLE 1.18: Layer ID: The Layer ID enumeration is the type for the Layer member of CPacketKey
R/W Property	IndexInLayer	Integer	Index of the packet key in its layer. It is a 1-based index.
Function	Constructor(layer, index)	CPacketKey	Construct a packet key with specified layer and index. Index is 1-based.
Function	Reset()	None	Reset the packet key to invalid state.

TABLE 1.16: Class CVSEConfig:

Type	Name	Result	Comments
R/W Property	TraverseOrder	TraverseOrder	Specifies the order in which VSE should send events to script. See below.
R/W Property	TraverseType	TraverseType	Specifies the type in which VSE should send events to script. See below.
R/W Property	AutoProgress	Boolean	Whether progress should be handled automatically by VSE engine or not.
R/W Property	SkipUnassociatedTraffic	Boolean	It is applicable only in Transaction mode. If it is TRUE, VSE skips un associated traffic during navigation. By default, it is TRUE in Transaction mode.
R/W Property	SkipNoneFrames	Boolean	It is applicable in both Linear and Transaction modes. If it is TRUE, VSE skips None frame events during navigating events. By default, it is FALSE in Transaction mode.
Function	SendPort(port)	None	port is of type VSEPort. See VSEPort enumerator.

Type	Name	Result	Comments
Function	`SendPort_2(boardindex, portindex)`	`None`	Boardindex is 1 based integer value which specifies board index of board. Portindex is port label that specified on board.
Function	`DontSendPort(port)`	`None`	port is of type VSEPort. See VSEPort enumerator.
Function	`DontSendPort_2(boardindex, portindex)`	`None`	Boardindex is 1 based integer value which specifies board index of board. Portindex is port label that specified on board.
Function	`SendItem(item_type)`	`None`	item_type is of type VSEItemType. See VSEItemType enumerator.
Function	`DontSendItem(item_type)`	`None`	item_type is of type VSEItemType. See VSEItemType enumerator.

TABLE 1.17: Class CTraceItem:

Type	Name	Result	Comments
Index	`[]`	`CTraceItem`	For given item_path string, it returns the corresponding TraceItem. See item_path below for more information.
R/O Property	`EventNumber`	`Integer`	Event number for this item (Link index). It is 1-based.
R/O Property	`PacketKeys`	`PacketKey[]`	Array of packet keys which this item contains.
R/O Property	`TimeStamp`	`Double`	Double precision floating point seconds since start which this item occurred.
R/O Property	`HasError`	`Bool`	Check to see if this item has any errors.
R/O Property	`ErrorList`	`TraceItemError[]`	List of all errors for this item. See TraceItemError for more information.
R/O Property	`StartBit`	`Integer`	Starting bit of this item within the its packet.
R/O Property	`Length`	`Integer`	Length of this item in bits.
R/O Property	`ErrorMessage`	`String`	Error message of this item.
R/O Property	`Data`	`TraceItemData`	Retrieves the data of this item. See TraceItemData for more information.
R/O Property	`Data2`	`TraceItemData`	Retrieves the data of this item. It returns “None” if field is empty.
R/O Property	`DecodedData`	`String`	Retrieves decoded string of current item’s data.
R/O Property	`Name`	`String`	This item’s name.
R/O Property	`Abbreviation`	`String`	This item’s abbreviation if available.
R/O Property	`Description`	`String`	Description string for this item.

Type	Name	Result	Comments
R/O Property	ByteOrder	ByteOrder	See ByteOrder for more information
R/O Property	BitOrder	BitOrder	See BitOrder for more information
R/O Property	ExchangeStatus	ExchangeState	See Table 1.12 for more information
R/O Property	PortNo	Integer	Retrieves one based port number of current item
R/O Property	PortNo_2	List	Retrieves 1 based port number of current item in a list. First item in list specifies board index, and second item specifies port number which is same as port label on the board.
R/O Property	PortAlias	String	Retrieves port alias name of current item
R/O Property	ItemPath	item_path	Retrieves path of item if item is a field.
R/O Property	ItemPathLongName	item_path	Retrieves path of item if item is a field. The path contains long(full) name of fields.
Function	GetItemByPath(item_path)	CTraceItem	Retrieves an item uniquely specified by item_path argument. item_path is of type String and is described below.
Function	GetItemByName(item_name)	CTraceItem	Retrieves the first sub item of this item with name equal to item_name parameter.
Function	GetAllItemsByName(item_name)	CTraceItem[]	Retrieves a list of all sub item of this item with name equal to item_name parameter. If user passes "*", it returns all items (fields) of event. You can also specify a field name with *, like TCP

Type	Name	Result	Comments
			Header.*, in this case it returns all subfields of TCP Header field.
Function	`GetMetaFieldData(logical_file_id)`	`LogicalField`	Retrieves TraceItemData of the logical field under this item specified by logical_file_id. See GELogicalField for more information.
Function	`GetMetaFieldDataList(logical_file_id)`	`LogicalField`	Retrieves list of TraceItemData of the logical field under this item specified by logical_file_id. See GELogicalField for more information more information. This function is applicable for Multi PDU protocols like iSCSI, NVMe and SMP. When a frame contains multi command PDU, you can get value of logical fields that are related to each of PDUs like RESPONSE_TIME and PENDING_EXCHANGES.
Function	`SetBookmark(name, desc)`	`None`	Creates bookmark at this item
Function	`GetBookmark()`	`CTraceBookmark`	Retrieves bookmark from this item. *TABLE 1.19: Class CTraceBookmark*
Function	`GetColumnData(Column_Name)`	`String`	Retrieves the column value of the current item based on the Spreadsheet view Column Name.

TABLE 1.18: Layer ID:

The Layer ID enumeration is the type for the Layer member of CPacketKey

Type	Name	Result	Comments
Enum Item	LAYER_LINK	Integer 32	Specifies Link layer
Enum Item	LAYER_SEQUENCE	Integer 32	Specifies transport layer
Enum Item	LAYER_SCSI_CMD	Integer 32	Specifies SCSI command layer
Enum Item	LAYER_LS_CMD	Integer 32	Specifies ELS command layer
Enum Item	LAYER_SW_CMD	Integer 32	Specifies switch command layer
Enum Item	LAYER_GS_CMD	Integer 32	Specifies GS command layer
Enum Item	LAYER_FICON_CMD	Integer 32	Specifies FICON command layer
Enum Item	LAYER_NVME_CMD	Integer 32	Specifies NVMe layer
Enum Item	LAYER_MAD_CMD	Integer 32	Specifies MAD layer
Enum Item	LAYER_ISCSI_CMD	Integer 32	Specifies iSCSI layer
Enum Item	LAYER_NVME_LS_CMD	Integer 32	Specifies NVMe Link Service layer
Enum Item	LAYER_SMB_CMD	Integer 32	Specifies SMB layer
Enum Item	LAYER_MPA_CMD	Integer 32	Specifies MPA layer
Enum Item	LAYER_IWARP_CMD	Integer 32	Specifies iWarp layer

TABLE 1.19: Class CTraceBookmark:

Type	Name	Result	Comments
R/W Property	Name	String	Specifies name of bookmark.
R/W Property	Description	String	Specifies description of bookmark.

Type	Name	Result	Comments
R/W Property	`PacketKey`	`CPacketKey`	Specifies packet key of packet. See Table 1.15

1.3.4 Types

Type TraceItemData:

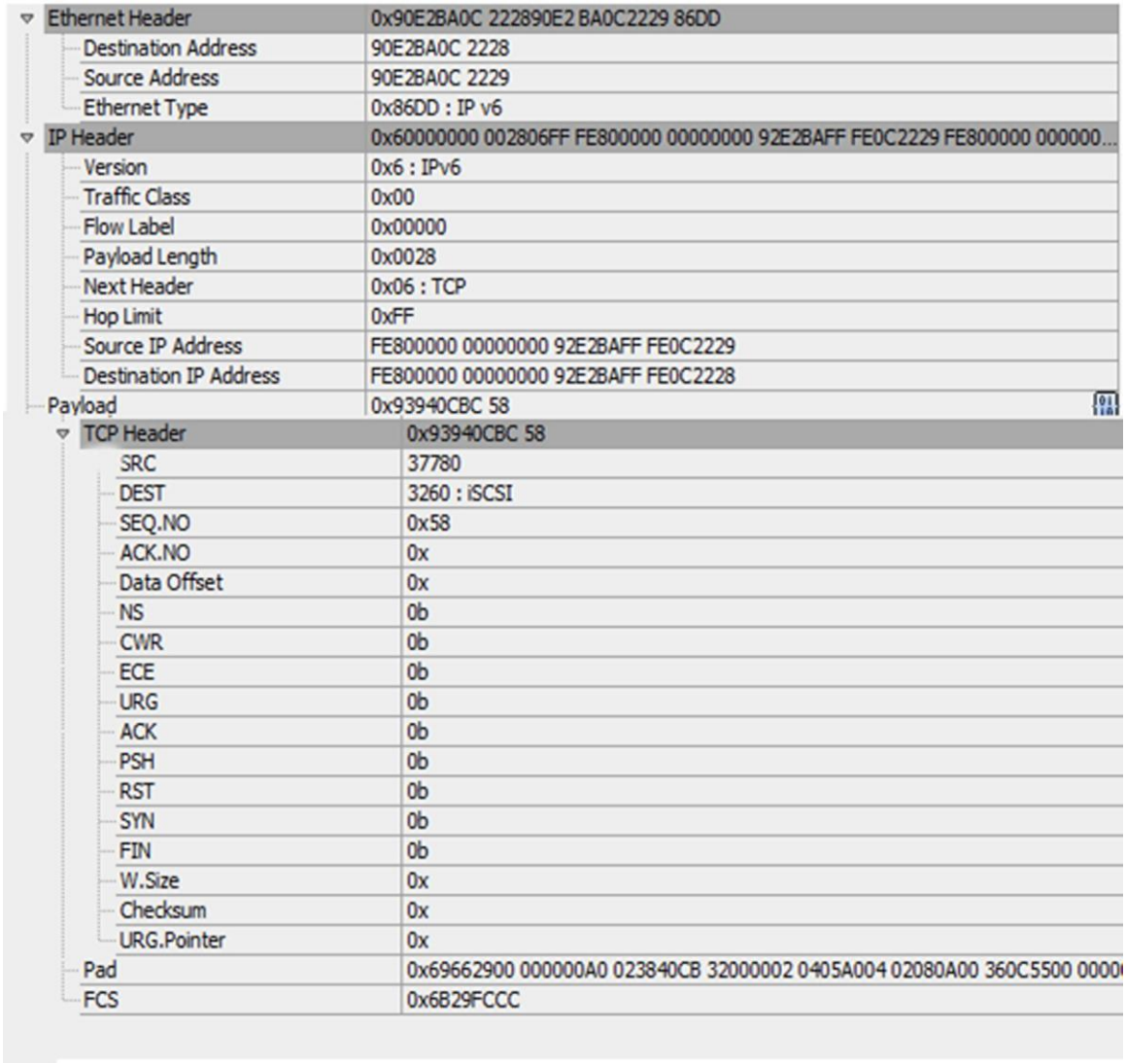
TraceItemData is a variant type consists of either Integer(8/16/32/64), String, floating point or `TraceItemBytes`. Actual type of the variant will be decided by engine at runtime depending on the operation which is being done.

Type TraceItemBytes:

Array of bytes. Each byte is accessible through indexing [] operator.

item_path string:

A string which can uniquely identify a field within an item. This string consists of field_names separated by period character. If a field is repeated in a given context, it is going to be treated as an array. Thus the user should specify which index he/she wants by specifying the index number in the square brackets. For example, consider the following packet which is presented by merging TCP header directly into the Ethernet payload:



Ethernet Header	0x90E2BA0C 222890E2 BA0C2229 86DD
Destination Address	90E2BA0C 2228
Source Address	90E2BA0C 2229
Ethernet Type	0x86DD : IP v6
IP Header	0x60000000 002806FF FE800000 00000000 92E2BAFF FE0C2229 FE800000 000000...
Version	0x6 : IPv6
Traffic Class	0x00
Flow Label	0x00000
Payload Length	0x0028
Next Header	0x06 : TCP
Hop Limit	0xFF
Source IP Address	FE800000 00000000 92E2BAFF FE0C2229
Destination IP Address	FE800000 00000000 92E2BAFF FE0C2228
Payload	0x93940CBC 58
TCP Header	0x93940CBC 58
SRC	37780
DEST	3260 : iSCSI
SEQ.NO	0x58
ACK.NO	0x
Data Offset	0x
NS	0b
CWR	0b
ECE	0b
URG	0b
ACK	0b
PSH	0b
RST	0b
SYN	0b
FIN	0b
W.Size	0x
Checksum	0x
URG.Pointer	0x
Pad	0x69662900 000000A0 023840CB 32000002 0405A004 02080A00 360C5500 0000
FCS	0x6B29FCCC

Here is a list of sample itempath in the above example:

- "Payload.TCP Header.SRC"
- "Ethernet Header.Source Address"

Note: If for example we had 2 consequent TCP Headers in the above example the first item would be:

```
"Payload.TCP Header[0].SRC"
```

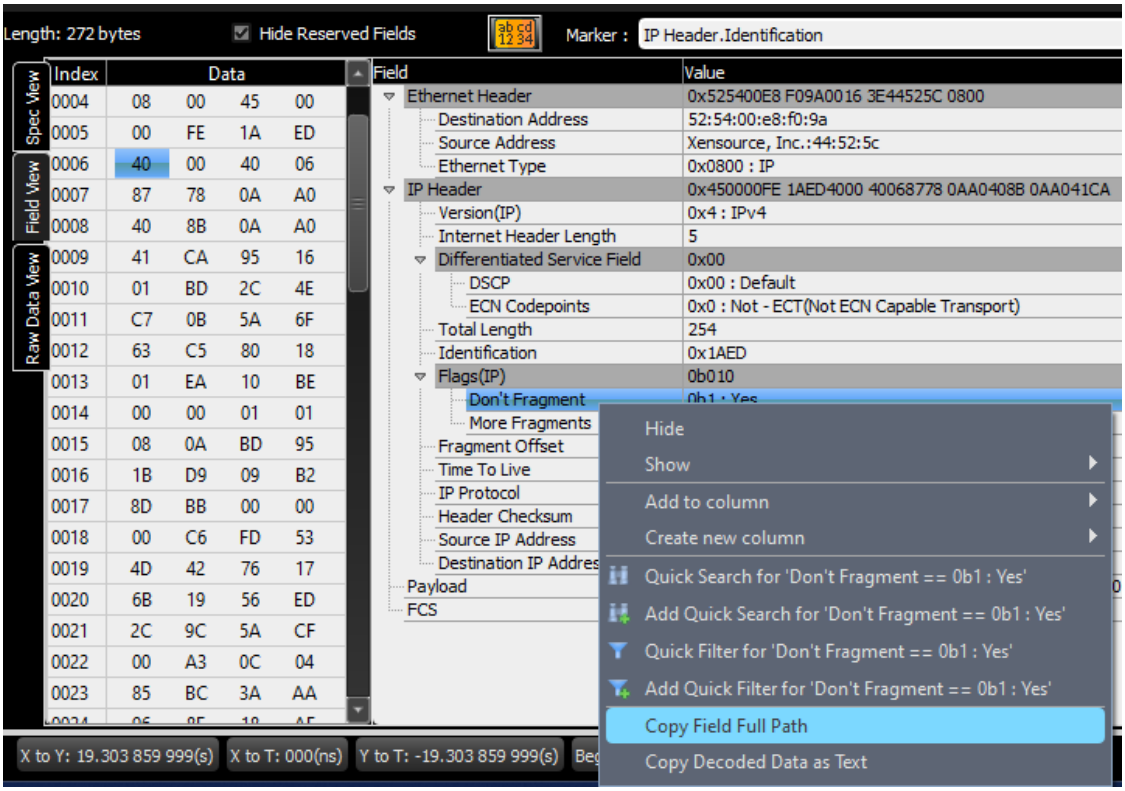
Period and square bracket characters in the field names can be represented by using escape sequences. An escape sequence is initiated with back-slash character. Character next to back-slash is considered as escaped character. Currently supported escape sequences are "\\", "\.", "\[", abd "\]"

without quotation marks. Using any escaped characters other than those mentioned will result in an error.

There are shortcuts for referencing field names, depending on which pane of the Frame Inspector view they are displayed in:

Pane	Shortcut
First Pane	“<Field Name>”
Second Pane	“.<Field Name>”
Third Pane	“..<Field Name>”

Note: User can copy an item_path string from inspector viewer in the Net Protocol Suite application. To do so, right click on any field in the Inspector View and click “Copy Field Full Path” as shown in the following figure. In this example, it will copy “**IP Header.Flags(IP).Don't Fragment**” to the clipboard and it can be used in VSE.



Appendix A

How to Contact Teledyne LeCroy

Send e-mail to Support	psgsupport@teledyne.com
Contact support	teledynelecroy.com/support/contact
Visit Teledyne LeCroy's web site	teledynelecroy.com
Tell Teledyne LeCroy	Report a problem to Teledyne LeCroy Support via e-mail by selecting **Help > Tell Teledyne LeCroy** from the application toolbar. This requires that an e-mail client be installed and configured on the host machine.

Appendix B

Examples of VSE Scripts

This appendix to the Net Protocol Suite VSE API Reference Manual shows a number of examples of VSE script files (located in the installation directory: C:\Users\Public\Documents\LeCroy\Net Protocol Suite\Examples\VSE).

B.1. Example Script 1: Custom Progress

```
# This example disables VSE's auto progress and report the reverse progress manually

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

total_items = Trace.ItemCount

# This callback will be called for each event in the trace
def OnEvent(item) :
    global total_items
    Output.SetProgress(0, total_items, total_items - item.EventNumber)

    # Continue traversing
    return True

# Create a VSE configuration to travese all ethernet items
config = CVSEConfig()
# Turn off auto progress
config.AutoProgress = False

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# Set the result to success
Output.Result = VSEResult.Success
```

B.2. Example Script 2: Ethernet Types

```
# This example counts all ethernet types in the trace and shows the number of
# each type at the end.

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

ethernet_types = dict()

# This callback will be called for each event in the trace
def OnEvent(item) :
    global ethernet_types

    type_item = item["Ethernet Header.Ethernet Type"]

    if type_item :
        type = type_item.Data
        if type in ethernet_types :
            current_type = ethernet_types[type]
            current_type[0] += 1
        else :
            ethernet_types[type] = [1, type_item.DecodedData]

    # Continue traversing
    return True

# Create a VSE configuration to travese all ethernet items
config = CVSEConfig()
config.TraverseOrder = TraceTraverseOrder.Linear
config.TraverseType = TraceTraverseType.Event
# Send no items but ethernet
config.SendItem(VSEItemType.NoneExcept)
config.SendItem(GEVSEItemType.Ethernet)

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# Print out the results
for k, v in ethernet_types.items():
    Output.AppendLog("Number of " + v[1] + " is: " + str(v[0]) + "\r\n")

# Set the result to success
Output.Result = VSEResult.Success
```

B.3. Example Script 3: Failure on Error

```
# This example traverse on all of the trace items and report failure as
# soon as it reaches the first protocol error

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

# This callback will be called for each event in the trace
def OnEvent(item) :
    if item.HasError :
        Output.AppendLog("Item number " + str(item.EventNumber) + " has error.")
        Output.Result = VSEResult.Failure
        # Stop traversing
        return False

    # Continue traversing
    return True

# Process VSE with default configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(CVSEConfig(), OnEvent)

# If the result is not already set then it means no errors found so set it to success.
if Output.Result == VSEResult.NoResult :
    Output.Result = VSEResult.Success
```

B.4. Example Script 4: FCP Frame Count

```
# This example counts all FCP Frames in the trace and shows the number of
# each type at the end.

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

fc_info = dict()

# This callback will be called for each event in the trace
def OnEvent(item) :
    global fc_info

    rctl_item = item["Frame Header.R_CTL"]
    type_item = item["Frame Header.TYPE"]

    if rctl_item and type_item :
        key = rctl_item.Data * 256 + type_item.Data
        if key in fc_info :
            current_info = fc_info[key]
            current_info[0] += 1
        else :
            fc_info[key] = [1, rctl_item.DecodedData]

    # Continue traversing
    return True

# Create a VSE configuration to travese all FC items
config = CVSEConfig()
config.TraverseOrder = TraceTraverseOrder.Linear
config.TraverseType = TraceTraverseType.Event
# Send no items but FC
config.SendItem(VSEItemType.NoneExcept)
config.SendItem(GEVSEItemType.FC)

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# Shows how many of each FCP frame type found in the trace.
for k, v in fc_info.items():
    Output.AppendLog("Number of " + v[1] + " is: " + str(v[0]))

# Set the result to success
Output.Result = VSEResult.Success
```

B.5. Example Script 5: GE Errors Count

```
# This example count all protocol errors in the trace and print out the number of
# each error type. It also shows the total number of events with protocol error.
Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")
# Convenient dictionary to translate protocol errors to string
GEErrors = {
    GETraceItemError.DELIMITER_ERROR: 'DELIMITER_ERROR',
    GETraceItemError.FRAME_LENGTH_ERROR: 'FRAME_LENGTH_ERROR',
    GETraceItemError.ETHERNET_CRC_ERROR: 'ETHERNET_CRC_ERROR',
    GETraceItemError.FC_CRC_ERROR: 'FC_CRC_ERROR',
    GETraceItemError.BLOCK_TYPE_ERROR: 'BLOCK_TYPE_ERROR',
    GETraceItemError.ORDER_SET_ERROR: 'ORDER_SET_ERROR',
    GETraceItemError.ALIGNMENT_ERROR: 'ALIGNMENT_ERROR',
    GETraceItemError.SYNC_HEADER_ERROR: 'SYNC_HEADER_ERROR',
    GETraceItemError.FEC_ERROR: 'FEC_ERROR',
    GETraceItemError.AUTO_NEG_FRAME_MARKER_ERROR: 'AUTO_NEG_FRAME_MARKER_ERROR',
    GETraceItemError.AUTO_NEG_MANCHESTER_ERROR: 'AUTO_NEG_MANCHESTER_ERROR',
    GETraceItemError.MARKER_INTERVAL_ERROR: 'MARKER_INTERVAL_ERROR',
    GETraceItemError.TRAINING_FRAME_MARKER_ERROR: 'TRAINING_FRAME_MARKER_ERROR',
    GETraceItemError.TRAINING_MANCHESTER_ERROR: 'TRAINING_MANCHESTER_ERROR',
    GETraceItemError.CHECKSUM_ERROR: 'CHECKSUM_ERROR',
    GETraceItemError.ECN_ERROR: 'ECN_ERROR'
}
errors_info = dict()
errors_count = 0

# This callback will be called for each event in the trace
def OnEvent(item) :
    global errors_info
    global errors_count
    if item.HasError :
        errors_count += 1
        for e in item.ErrorList :
            if e in errors_info :
                errors_info[e] += 1;
            else :
                errors_info[e] = 1
    # Continue traversing
    return True

# Create a VSE configuration to travese all ethernet items
config = CVSEConfig()
config.TraverseOrder = TraceTraverseOrder.Linear
config.TraverseType = TraceTraverseType.Event
# Send no items but ethernet
config.SendItem(VSEItemType.NoneExcept)
config.SendItem(GEVSEItemType.Ethernet)
config.SendItem(GEVSEItemType.Primitive)

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# Print out the results
Output.AppendLog("Total number of events with error: " + str(errors_count))
for k, v in errors_info.items():
    Output.AppendLog("Number of " + GEErrors[k] + " is: " + str(v))

# Set the result to success
Output.Result = VSEResult.Success
```

B.6. Example Script 6: Success On Read10

```
# This example reports success as soon as it finds the first item with
# SCSI opcode Read10 on all ports except port 1

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

# This callback will be called for each event in the trace
def OnEvent(item) :
    fc_item = item["Data.FCP_CDB.Operation Code"]
    if fc_item and fc_item.Data == 0x28 :
        Output.Result = VSEResult.Success;
        # Stop traversing
        return False

    ethernet_item = item["Payload.iSCSI.CDB.Operation Code"]
    if ethernet_item and ethernet_item.Data == 0x28 :
        Output.Result = VSEResult.Success;
        # Stop traversing
        return False

    # Continue traversing
    return True

# Create a VSE configuration to travese all ethernet items
config = CVSEConfig()
# Send all ports but port 1
config.SendPort(VSEPort.AllExcept)
config.DontSendPort(1)

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# If the result is not already set then it means no Read10 found so set it to failure.
if Output.Result == VSEResult.NoResult :
    Output.Result = VSEResult.Failure
```

B.7. Example Script 6: Success On Read10 (Using DontSendPort_2)

```
# This example reports success as soon as it finds the first item with
# SCSI opcode Read10 on all ports except port 1

Output.AppendLog(Trace.Name + " has " + str(Trace.ItemCount) + " items.")

# This callback will be called for each event in the trace
def OnEvent(item) :
    fc_item = item["Data.FCP_CDB.Operation Code"]
    if fc_item and fc_item.Data == 0x28 :
        Output.Result = VSEResult.Success;
        # Stop traversing
        return False

    ethernet_item = item["Payload.iSCSI.CDB.Operation Code"]
    if ethernet_item and ethernet_item.Data == 0x28 :
        Output.Result = VSEResult.Success;
        # Stop traversing
        return False

    # Continue traversing
    return True

# Create a VSE configuration to travese all ethernet items
config = CVSEConfig()
# Send all ports but port 1
config.SendPort(VSEPort.AllExcept)
config.DontSendPort_2(1, 1)

# Process VSE with the specified configurations and call OnEvent callback
# for each item.
Trace.ProcessVSE(config, OnEvent)

# If the result is not already set then it means no Read10 found so set it to failure.
if Output.Result == VSEResult.NoResult :
    Output.Result = VSEResult.Failure
```