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

PE*Trainer*™

Scripting Language

Reference Manual

Manual Version 1.7

For PE*Tracer*™ Software Version 5.00

June 2006

Document Disclaimer

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

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

Trademarks and Servicemarks

CATC Trace, PETrainer EML, PETrainer ML, PETracer EML, PETracer ML, PETracer, and *BusEngine* are trademarks of LeCroy.

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

All other trademarks are property of their respective companies.

Copyright

Version

This is version 1.7 the PE*Trainer* Scripting Language Reference Manual. This manual applies to PE*Tracer* software version 5.00 and higher.

Contents

1 Introduction

This manual describes the scripting language used to create traffic generation files for PE*Trainer*™.

1.1 Software Version

This document is for: PE*Tracer*™ software 5.00.

1.2 Support Resources

As new functionalities are added, not all of them are supported by older versions of the PE*Tracer* software. For newer releases of the analyzer's software, please refer to LeCroy's web site: http://www.lecroy.com/

2 Syntax

PE*Trainer*™ Script files consist of the statements with the following format:

```
COMMAND = MODIFIER {
      PARAM1 = VALUE1
      …
      PARAMn = VALUEn
}
```

See the list of all commands with all applicable modifiers on page 3. For some commands the list of the parameters is optional.

All literals are not case sensitive.

All default values are zeros unless otherwise noted.

Integer literals represent numeric values with no fractions or decimal points.

Hexadecimal, decimal, and binary notation are supported:

- Hexadecimal numbers must be preceded by 0x: 0x2A, 0x54, 0xFFFFFF01
- Decimal numbers are written as usual: 24, 1256, 2
- Binary numbers are denoted with 0b: 0b01101100, 0b01, 0b100000

It is possible to use expressions, for example, (i - 239). See Page 56 for more examples.

String literals are surrounded by double quotes.

Array data types are represented by integer or string literals surrounded by “(“ and “)” characters, and separated by a comma “,”. For example, (2,23,4).

Single-line comments are supported and should be preceded by a semicolon “;”.

Multi-line comments are also supported. Multi-line comments begin with a “/*” combination, and end with the reverse “*/” combination.

3 Command List

COMMAND	MODIFIERS	Comment
Packet	TLP, DLLP, OrderedSet, Raw, <TemplateName>	Sends a packet
Idle	<# of ns>	Sends idle symbols (D0.0)
Link	L0, L1, L0s, HotReset, Disabled, Recovery, Detect, LTSSMOff, InitFC	Sets a link condition
Config	General, FCTx, FCRx, TLP, AckNak, Transactions, Link, Definitions	Configures the PE*Trainer*™
Wait	TLP, DLLP, Error, LinkCondition, BOB, Payload, User	Waits for the condition specified
Include	<Include file path>	Includes a PE*Trainer* script file
Branch	TLP, DLLP, Error, Link, BOB, Payload, User	Enables/disables an interrupt for the specified condition
Proc	Begin, End	Declares the procedure to be used in a branch statement
Loop	Begin, End	Creates a PE*Trainer* loop
Repeat	Begin, End	Repeats traffic some number of times
Template	TLP, DLLP, OrderedSet, Raw, <TemplateName>	Creates a template for a packet that can be used in the Packet command
AddressSpace	Read, Write	Reads/Writes address space

4 Packet Command

This command initiates transmission of a specified packet on the bus.

4.1 Packet = TLP

This command initiates transmission of TLP packet on the bus. The parameters of the **Packet = TLP** command cover all the fields in the TLP header: TLP Payload, PSN (Packet Sequence Number), ECRC, and LCRC. Reserved fields can be set with the **RawData** parameter.

Parameter	Values	Default Value	Comment
PSN	0:4095, Incr	0	When **Incr** is specified, the PSN for the current TLP is assigned as the PSN of the previously sent TLP incremented by 1. When the PSN is generated automatically (see the **AutoSeqNumber** parameter, Page 35), this parameter has no effect.
TLPType	MRd32 MRdLk32 MWr32 MRd64 MRdLk64 MWr64 IoRd IoWr CfgRd0 CfgWr0 CfgRd1 CfgWr1 Msg MsgD Cpl CplLk CplD CplDLk	0	Sets the **Fmt** (bits 6:5 of byte 0 in the TLP header) and **Type** (bits 4:0 of byte 0 in the TLP header) fields in the TLP header. Also, this field can be specified as a direct numeric value that would specify bits 6:0 of byte 0 in the TLP header.
TC	0:7	0	**Traffic Class**: bits 6:4 of byte 1 in the TLP header
TD	0:1	0	Bit 7 of byte 2 in the TLP header: 1 indicates presence of TLP digest in the form of a single DW at the end of the TLP.
EP	0:1	0	Bit 6 of byte 2 in the TLP header: indicates the TLP is poisoned.
Snoop	0:1	0	Bit 4 of byte 2 in the TLP header: 0 indicates that hardware enforced cache coherency is expected. 1 indicates that hardware enforced cache coherency is not expected.

Ordering	0:1	0	Bit 5 of byte 2 of TLP header: 0 indicates PCI Strongly Ordered Model. 1 indicates PCI-X Relaxed Ordering Model.
Length	0:1023	0	Length of data payload in DWORDs. If not specified, this field is 1 for all read requests and calculated according to the actual payload for write requests.
Tag	0:255	0	Byte 6 of the TLP Header for Memory, IO, and Configuration TLP packets. Byte 10 for Completion TLP packets. When Tags are generated automatically (see the **TagGeneration** parameter, Page 35), this parameter has no effect for Memory, IO, and Configuration TLP packets.
RequesterID	(XX:XX:X) or direct value	0	Bytes 4-5 of the TLP Header for Memory, IO, and Configuration TLP packets. Bytes 8-9 for Completion TLP packets. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
ECRC	0x00000000: 0xFFFFFFFF	Calculated automatically	When not specified, the PE*Tracer*™ software automatically calculates the ECRC. (TD field has to be specified.)
LCRC	0x00000000: 0xFFFFFFFF	Calculated automatically	When not specified, the PE*Tracer* software automatically calculates the LCRC. When LCRC is generated automatically by the PE*Trainer*™ hardware (see the **AutoLCRC** parameter, Page 35), this parameter has no effect.
Payload	(XXXX,XXXX,…) Incr Random Zeros Ones		Specified as the array of DWORDs in hexadecimal format (Big Endian). The **Payload** parameter applies only to TLP packets with data. **Incr**: Specifies a payload as the sequence (0, 1, …'**Length**'). **Random**: Specifies a random payload. **Zeros**: Specifies a payload of all zeros. **Ones**: Specifies a payload of all ones. Note: When **Incr, Random, Zeros,** and **Ones** are used, the **Length** parameter must be specified before the payload. **Payload** can be specified for Memory, IO, Configuration writes, and Completion with Data TLP packets.

Field[<start>:<end>] Field[<pos>]			The arbitrary TLP Header field could be specified by using **Field** parameter. **Start, end**, and **pos** are bit positions from the beginning of TLP Header. Position 0 corresponds to the Most Significant Bit of the first byte of TLP Header. Position 95 for 3 DWORD header (and position 127 for 4 DWORD header) correspond to the Least Significant Bit of the last byte of TLP Header. Fields are limited by 32 bit values. Use **Field[<start>:<end>]** syntax to specify multi bit field. Use **Field[<pos>]** to specify single bit field.
RawData@<start>			Inserts raw data symbols at **<start>** byte position from the beginning of the TLP. See the **Packet = Raw** description, Page 27, for possible raw data formats.
Count	1:65535	1	Repeats this packet by the number of times specified.

Example 1:

Read one DWORD of data from address 0x1000.
Length parameter is not specified, so the default value of 1 is used.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      PSN = 0
      TLPType = MRd32
      Address = 0x1000
}
```

Example 2:

Read 32 DWORDs of data starting from address 0x1000.
PSN would accept values 0 for first TLP and 1 for second TLP.
TC, EP, Ordering, and **Snoop** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.
ECRC is not specified, so the **ECRC** is calculated by software.

```
Packet = TLP {
      PSN = Incr
      TLPType = MRd32
      Tag = 0
      Address = 0x1000
      TD = 1
      FirstDwBe = 0xF
      Length = 16
}
Packet = TLP {
      PSN = Incr
      TLPType = MRd32
      Tag = 1
      Address = 0x1010
      TD = 1
      FirstDwBe = 0xF
      Length = 16
}
```

Example 3:

This example does not specify **PSN, Tag,** and **LCRC**. Those values are calculated automatically by the PE*Trainer* hardware (see more on **Config = TLP** command, Page 36).

```
Config = TLP {
      AutoSeqNumber = Yes
      AutoLCRC = Yes
      TagGeneration = Default
}

Packet = TLP {
      TLPType = MRd32
      Address = 0x1010
      TD = 1
      Length = 1
}
```

Example 4:

This example shows how to specify a reserved field in the TLP header using the **RawData** parameter (see more on the **RawData** parameter, Page 27).

```
Packet = TLP {
      TLPType = MRd32
      Address = 0x1010
      RawData@4 = ( D1 )
}
```

Example 5:

This example shows how to specify reserved fields in the TLP header using the **Field** parameter:

```
Packet=TLP {
      TLPType=CfgRd0
      Register = 0x34
      Length = 1
      FirstDwBe = 0xF
      Field[0] = 0x1
      Field[8] = 0x1
      Field[12:15] = 0xF
      Field[20:21] = 0x3
      Field[80:83] = 0xF
}
```

Example 6:

This example shows how to specify the TLP type directly. Any invalid TLP type can be generated with this method.

```
Packet = TLP {
      TLPType = 0x4F
}
```

Example 7:

Repeat this TLP packet 64 times.

```
Packet = TLP {
      TLPType = MRd32
      Address = 0x1000
      Count = 64
}
```

4.1.1 TLPType = MRd32, MRdlk32, MWr32

Parameter	Value	Default	Comment
LastDwBe	0:15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0:15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
Address	0x00000000: 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.

Example 1:

This example shows how to send a 32-bit Memory Write TLP.
The **Length** field is not specified, so it would be calculated by software. (**Length = 4** would be used.)
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      TLPType = MWr32
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x1000
      Payload = ( 0x2, 0x4, 0x6, 0x8 )
}
```

Example 2:

This example shows how to send a 32-bit Memory Write TLP. This command would generate a random payload of 1024 DWORDs.

```
Packet = TLP {
      TLPType = MWr32
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x1000
      Length = 0            ; 0 means 1024 DWORDs of payload
      Payload = Random
}
```

4.1.2 TLPType = MRd64, MRdLk64, MWr64

Parameter	Value	Default	Comment
LastDwBe	0:15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0:15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
AddressLo	0x00000000: 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.
AddressHi	0x00000000: 0xFFFFFFFF	0	Bytes 12-15 in the TLP header.

Example 1:

This example shows how to send a 64-bit Memory Write TLP.
Length parameter is set to 3 intentionally in order to generate a TLP with incorrect length.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software.

```
Packet = TLP {
      TLPType = MWr64
      LastDwBe = 0xF
      FirstDwBe = 0xF
      AddressLo = 0x1000
      AddressHi = 0x60000000
      Payload = ( 0x2, 0x4, 0x6, 0x8, 0x2, 0x4, 0x6, 0x8 )
      Length = 3
}
```

4.1.3 TLPType = IoRd, IoWr

Parameter	Value	Default	Comment
LastDwBe	0:15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0:15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
Address	0x00000000: 0xFFFFFFFF	0	Bytes 8-11 in the TLP header.

Example 1:

Read one DWORD of data from address 0x1000 of the IO address space.
Length parameter is not specified, so the default value of 1 is used.
TC, TD, EP, Ordering, Snoop, and **Tag** parameters are not specified, so the default value of 0 is used.
LCRC is not specified, so the **LCRC** is calculated by software

```
Packet = TLP {
      TLPType = IoRd
      Address = 0x1000
}
```

4.1.4 TLPType = Cfgrd0, Cfgwr0, Cfgrd1, Cfgwr1

Parameter	Value	Default	Comment
LastDwBe	0:15	0	Byte 7 in the TLP header. See rules for **Last DW BE** in the PCI Express Specification.
FirstDwBe	0:15	0	Byte 7 in the TLP header. See rules for **1st DW BE** in the PCI Express Specification.
DeviceID	(XX:XX:X) or direct value	0	Bytes 8-9 in the TLP header. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
Register		0	Bytes 10-11 in the TLP header.

Example 1:

This example reads the Capability Pointer from the device's configuration space (**Bus Number 0, Device Number 2, Function Number 4**).

```
Packet = TLP {
      TLPType = CfgRd0
      DeviceId = (0:2:4)
      Register = 0x34
      Length = 1
      FirstDwBe = 0x1
}
```

Example 2:

This example writes to the Command Register of the device's configuration space (**Bus Number 0, Device Number 0, Function Number 1**).

```
Packet = TLP {
      TLPType = CfgWr0
      DeviceId = 1
      Register = 0x04
      Length = 1
      FirstDwBe = 0x3
      Payload = ( 0x03000000 )
}
```

4.1.5 TLPType = Msg, Msgd

Parameter	Value	Default	Comment
MessageRoute	ToRootComplex ByAddress ByID FromRootComplex Local Gather	ToRootComplex	**MessageRoute** affects the **Type** field of TLP header. (Bits 2:0).
MessageCode	Assert_INTA Assert_INTB Assert_INTC Assert_INTD Deassert_INTA Deassert_INTB Deassert_INTC Deassert_INTD PM_Active_State_Nak PM_PME PME_Turn_Off PME_TO_Ack ERR_COR ERR_NONFATAL ERR_FATAL Unlock Set_Slot_Power_Limit Vendor_Defined_Type0 Vendor_Defined_Type1 Attention_Indicator_On Attention_Indicator_Blink Attention_Indicator_Off Power_Indicator_On Power_Indicator_Blink Power_Indicator_Off Attention_Button_Pressed Direct numeric values can also be used.	0	Byte 7 in the TLP Header
AddressHi	0x00000000: 0xFFFFFFFF	0	Used only if **MessageRoute=ByAddress**
AddressLo	0x00000000: 0xFFFFFFFF	0	Used only if **MessageRoute=ByAddress**
DeviceID	(XX:XX:X) or direct value	0	Used only if **MessageRoute=ById**. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)

Example 1:

This example shows how to send a **PME_Turn_Off** Power Management Message while emulating the Root Complex.

```
Packet = TLP {
      TLPType = Msg
      MessageCode = PME_Turn_Off
      MessageRoute = FromRootComplex
}
```

Example 2:

This example shows how to send a **Vendor_Defined_Type0** Vendor Defined Message to the function 1 of device 1 on bus 0.

```
Packet = TLP {
      TLPType = Msg
      MessageCode = Vendor_Defined_Type0
      MessageRoute = ByID
      DeviceID = (0:1:1)
}
```

4.1.6 TLPType = Cpl, CplLk, CplD, CplDLk

Parameter	Value	Default	Comment
CompleterId	(XX:XX:X) or direct value	0	Identifies the Completer. This parameter can be set in the following format: (**BusNumber : DeviceNumber : FunctionNumber**)
ComplStatus	SC UR CRS CA	SC	Indicates the completion status.
BCM	0:1	0	**Byte Count Modified**: Must not be set by PCI Express Completers and may only be set by PCI-X completers. Indicates that the Byte Count field reports the size of just the first packet instead of the entire remaining byte count.
ByteCount	0:4095	0	Remaining byte count for the request
LowerAddr	0:63	0	Lower byte address for the starting byte of the completion

Example 1:

This example shows how to send a Completion TLP. This Completion TLP returns Unsupported Request (UR) status.

Requester is Function 0 of Device 0 on Bus 0.

Completer is Function 0 of Device 1 on Bus 0.

This completes the TLP request with Tag Number 4.

```
Packet = TLP {
      TLPType = Cpl
      RequesterId = (0:0:0)
      CompleterId = (0:1:0)
      Tag=4
      ComplStatus = UR
}
```

Example 2:

This example shows how to send a Completion with Data TLP. This Completion TLP returns Successful Completion (SC) status.

Requester is Function 0 of Device 0 on Bus 0.

Completer is Function 0 of Device 1 on Bus 0.

This completes the TLP request with Tag Number 4.

This is the last Completion of the Split Transaction since ByteCount field is equal to the number of bytes transferred and BCM is not set.

```
Packet = TLP {
      TLPType = CplD
      RequesterId = (0:0:0)
      CompleterId = (0:1:0)
      Tag=4
      ComplStatus = SC
      ByteCount = 32
      Payload = ( 0x00000001, 0x00000002, 0x00000003, 0x00000004,
                  0x00000005, 0x00000006, 0x00000007, 0x00000008 )
}
```

4.2 Packet = DLLP

This command initiates transmission of DLLP packets on the bus.
Parameters for the **Packet = DLP** command cover all the fields in a DLLP.
Reserved fields can be set using the **RawData** parameter.

Parameter	Values	Default	Comment
DLLPType	Ack Nak InitFC1_P InitFC1_NP InitFC1_Cpl InitFC2_P InitFC2_NP InitFC2_Cpl UpdateFC_P UpdateFC_NP UpdateFC_Cpl PM_Enter_L1 PM_Enter_L23 PM_Active_State_Request_L1 PM_Request_Ack Vendor		First byte in the DLLP
CRC	0: 65535	Automatically calculated	Bytes 4-5 in the DLLP. When not specified, it is calculated automatically.
Field[<start>:<end>] Field[<pos>]			The arbitrary DLLP field could be specified by using **Field** parameter. **Start, end**, and **pos** are bit positions from the beginning of DLLP. Position 0 corresponds to the Most Significant Bit of the first byte of DLLP. Position 31 corresponds to the Least Significant Bit of the last byte of DLLP. Use **Field[<start>:<end>]** syntax to specify multi bit field. Use **Field[<pos>]** to specify single bit field.
RawData@<start>			Inserts raw data symbols at **<start>** byte position from the beginning of the DLLP. See **Packet = Raw** description (page 27) for possible raw data formats.
Count	1: 65535	1	Repeats this packet by the number of times specified.

Example 1:

This example shows how to send a **PM_Active_State_Request_L1** power management DLLP. This DLLP would be sent 132 times.

The DLLP's **CRC** is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = PM_Active_State_Request_L1
      Count = 132
}
```

Example 2:

This example shows how to send a DLLP with an incorrect CRC.

```
Packet = DLLP {
      DLLPType = PM_Enter_L1
      CRC = 0x1234
}
```

Example 3:

This example shows how to modify reserved fields in a DLLP using the **RawData** parameter. (See more on the **RawData** parameter, Page 27.)

```
Packet = DLLP {
      DLLPType = PM_Active_State_Request_L1
      RawData@3 = ( D11.1, D11.2 )
}
```

Example 4:

This example shows how to specify reserved fields in a DLLP using the **Field** parameter.

```
Packet = DLLP {
      DLLPType = Ack
      Field[8:19] = 0b101001000111
}
```

4.2.1 DLLPType = Ack, Nak

Parameter	Values	Default	Comment
AckNak_SeqNum	0:4095	0	Bytes 2-3 in the DLLP

Example 1:

This example acknowledges all TLP packets with a sequence number less than or equal to 120 and initiates retransmission of TLP packets with a sequence number more than 120. The DLLP's CRC is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = Ack
      AckNak_SeqNum = 120
}
```

4.2.2 DLLPType = InitFC1_P, InitFC1_NP, InitFC1_Cpl, InitFC2_P, InitFC2_NP, InitFC2_Cpl, UpdateFC_P, UpdateFC_NP, UpdateFC_Cpl

Parameter	Values	Default	Comment
VC_ID	0:7	0	Virtual Channel, bits 2:0 in the first byte of the DLLP
HdrFC	0:255	0	Contains the credit value for headers of the indicated type (P, NP, or Cpl)
DataFC	0:4095	0	Contains the credit value for payload Data of the indicated type (P, NP, or Cpl)

Example 1:

The following example initializes credits for VC 0 for posted TLP requests.
Credit value for headers is 0. Credit value for data payload is infinite.
The DLLP's **CRC** is calculated automatically since **CRC** is not specified.

```
Packet = DLLP {
      DLLPType = InitFc1_P
      VC_ID = 0
      HdrFC = 2
      DataFC = 0

}
```

4.2.3 DLLPType = Vendor

Parameter	Values	Default	Comment
Data	0x000000:0xFFFFFF	0	Vendor specific data, bytes 1-3 in the DLLP

Example 1:

```
Packet = DLLP {
      DLLPType = Vendor
      VendorSpecific = 0x010203
}
```

4.3 Packet = OrderedSet

This command initiates transmission of ordered set on the bus.

Parameter	Values	Default	Comment
SetType	TS1 TS2 FTS Pattern Idle Skip		
RawData@<start>			Inserts raw data symbols at **<start>** byte position from the beginning of the ordered set. See **Packet = Raw** (page 27) description for possible raw data formats.
Count	1: 65535	1	Repeats this packet by the number of times specified.

Example:

The following example sends 255 Fast Training Sequences.

```
Packet = OrderedSet {
      SetType = FTS
      Count = 255
}
```

4.3.1 SetType = TS1, TS2

Parameter	Values	Default	Comment
LinkNumber	0:255, PAD	PAD	Link Number within component
LaneNumber	0:31, PAD	PAD	Lane Number within Port
N_FTS	0:255	0	The number of fast training ordered sets required by the Receiver to obtain reliable bit and Symbol lock.
TrainingControl	(X,X,X,X)	(0,0,0,0)	Training control bits. The order of the bits is as follows: **(HotReset, DisableLink, Loopback, DisableScrambling)**
Identifier	(X,X,X…)	D10.2 for TS1 and D5.2 for TS2	Use the same format as in **Packet = Raw** (see page 27), with exception of 10-bit codes.

In x4, x8 or x16 configurations, the keys listed above apply to all lanes.

When you want to specify parameters for a particular lane, use the following format:

```
<key>@<lane_number> = <value>
```

Example 1:

The following example sends a TS1 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
TrainingControl bits are zeroes for all lanes.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
      N_FTS = 255
}
```

Example 2:

The following example sends a TS1 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber is 0 for all lanes.
LaneNumber are 3, 2, 1, 0 for lanes 0, 1, 2, 3, and PADs for all other lanes.
TrainingControl bits are zeroes for all lanes.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
      LinkNumber = 0
      LaneNumber@0 = 3
      LaneNumber@1 = 2
      LaneNumber@2 = 1
      LaneNumber@3 = 0
      N_FTS = 255
}
```

Example 3:

The following example sends a TS2 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
TrainingControl's **Disable Scrambling** bit is asserted on all lanes. All other **TrainingControl** bits are de-asserted.
Identifier symbols are (D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2 D10.2) for all lanes.

```
Packet = OrderedSet {
      SetType = TS1
      N_FTS = 255
      TrainingControl = (0,0,0,1)
}
```

Example 4:

The following example sends a TS2 ordered set.
N_FTS is equal to 255 for all lanes.
LinkNumber and **LaneNumber** are PADs (the default value) for all lanes.
All **TrainingControl** bits are de-asserted.
Identifier symbols are (D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.1, D5.2) for lane 2.
Identifier symbols are (D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2) for all other lanes.
This would send a corrupted TS2 ordered set, since the **Identifier** is incorrect for lane 2.

```
Packet = OrderedSet {
    SetType = TS2
    N_FTS = 255
    Identifier@2 = (D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2, D5.2,
D5.1, D5.2)
}
```

4.3.2 SetType = Skip

Parameter	Values	Default	Comment
SkipCount	0:5	3	Number of SKIP symbols to send after COMMA

Example 1:

This example sends a Skip ordered set. Comma followed by 3 SKIP symbols would be sent on each lane.

```
Packet = OrderedSet {
      SetType = Skip
}
```

Example 2:

This example sends a Skip ordered set. Comma followed by 2 SKIP symbols would be sent on each lane.

```
Packet = OrderedSet {
      SetType = Skip
      SkipCount = 2
}
```

4.4 Packet = Raw

This command initiates transmission of raw data on the bus.

Parameter	Values	Default	Comment
RawData	(X,X,X…)		Specifies the array of bytes or 10-bit symbols to send.
Count	1: 65535	1	Repeats packet specified number of times.

The elements of data can be specified in the following formats:

1) Symbols:

```
Packet = Raw
{
      RawData = ( K28.5, D21.5, K28.5, D10.2 )
}
```

2) Bytes in hexadecimal format with preceding K/D modifier:

```
Packet = Raw
{
      RawData = ( KBC, DB5, KBC, D4A )
}
```

3) In addition to generate fully qualified 10 bit symbols you can specify running disparity sign for each symbol:

```
Packet = Raw
{
      RawData = ( K28.5+, D21.5-, K28.5-, D10.2- )
}
```

4) Specify 10 bit symbols in binary, hex or decimal format:

```
Packet = Raw
{
  RawData = ( 0b0011111010, 0b1100111001, 0b0011111010, 0b1110000110 )
}
```

4.5 Packet = <TemplateName>

This command initiates transmission of the packet specified by the **Template** command (see page 59). User can override packet fields according to the template.

Example 1:

This sequence issues three 32-bit Memory read requests. The address field of TLP header would accept the values 0, 64, and 128. Every other field in the TLP header would accept the value from the packet template.

```
Template = TLP {
      Name = “TestPacket”
      Type = MRd32
      RequesterID = (1:0:0)
      Length = 64
      Address = 0
}


Packet = “TestPacket”
{
}

Packet = “TestPacket”
{
      Address = 64
}

Packet = “TestPacket”
{
      Address = 128
}
```

5 Idle Command

This command sends idle symbols (D0.0) for the time specified.

Example:

The following example sends two TLP packets separated by D0.0 symbols. The idle time between those TLP packets is 64 ns. Eight D0.0 symbols would be sent between TLP packets on each lane.

```
Packet = TLP {
      TLPType = MRd32
      Address = 0x1000
}

Idle = 64

Packet = TLP {
      TLPType = MRd32
      Address = 0x1000
}
```

6 Link Command

All of these commands, with the exception of **Link = InitFC**, are controls to the Link Training and Status State Machine (also known as the LTSSM). These commands are issued to the LTSSM to steer it to a particular state. This is not a means to force the Link state to a particular value. For instance, if the Script contains the **Link = L0** command, it is a request to bring the link to the L0 state. The LTSSM is responsible for managing all of the link training and all of the intermediate link states to accomplish this.

6.1 Link = L0

Transitions the link into the L0 state.

6.2 Link = L1

Transitions the link into the L1 (low power) state. Applies only in L0 state.

6.3 Link = L0s

Transitions the link into the L0s (low power) state. Applies only in L0 state.

6.4 Link = Disabled

Tells the LTSSM to move into the Disabled State. To get to this state, the LTSSM must either be in the Configuration State or the Recovery State. If the link is currently in the Detect state, and the **Link=Disabled** command is issued, it goes to Configuration first and then goes directly to Disabled. Once in the Disabled state, the LTSSM sends 16 TS1's with the Disable Link modifier bit set, followed by an electrical Idle ordered set, followed by electrical idle. To exit the Disabled state, simply set **Link=Detect** or **Link=L0**.

6.5 Link = HotReset

Tells the LTSSM to move into the HotReset State. To get to this state, the LTSSM must first be in the Recovery state. Once in the HotReset State, the LTSSM sends TS1 ordered sets with the HotReset modifier bit set. The LTSSM then goes to the Detect state automatically after 2 ms.

6.6 Link = Recovery

Transitions the link into the Recovery state. Applies only in L0, L0S, or L1 States.

6.7 Link = Detect

Tells the PE*Trainer*™ to immediately bring the Link down. In this state, the LTSSM drives all of the PCI Express lanes to electrical idle. Before the lanes go to electrical idle, a single electrical idle ordered set is transmitted. Applies while in any state.

6.8 Link = LTSSMOff

Disables the LTSSM. This essentially means that the PE*Trainer* is not responsible for managing the link state. Instead, the user is free to transmit ordered sets, DLLP's, and RAW packets blindly.

6.9 Link = InitFC

Starts the flow control initialization state machine.

6.10 Link = PERST

Sends a **PERST#** signal for the period specified.

Parameter	Values	Default	Comment
Duration	In ns (rounded to nearest 8)	1000	Duration of the PERST# signal

7 Config Command

This command configures the PE*Trainer*™.

7.1 Config = General

This command should precede any statement in a PE*Trainer* script file. There should be only one **Config = General** command in a PE*Trainer* script file. All **Config = General** commands from included files (see page 61) are ignored.

Parameter	Values	Default	Comment
AutoDetect	Yes, No	No	Automatically detect link parameters.
LinkWidth	1,4,8,16	4	Ignored in PE*Trainer* ML if AutoDetect is set.
DirectionRx	Upstream, Downstream	U	
DisableScrambleTx	Yes, No	No	Ignored if AutoDetect is set.
DisableDescrambleRx	Yes, No	No	Ignored if AutoDetect is set.
ReverseLanesTx	Yes, No	No	Ignored if AutoDetect is set.
ReverseLanesRx	Yes, No	No	Ignored if AutoDetect is set.
InvertPolarityTx	(X,X,X,X,…)		The array of 1/0 elements. The size of the array should match the link width.
InvertPolarityRx	(X,X,X,X,…)		The array of 1/0 elements. The size of the array should match the link width. Ignored if AutoDetect is set.
BaseSpec10	Yes, No	No	
SkewTx	(X,X,X,X,…)		The array of integer elements. The size of the array should match the link width. Measured in symbols, valid values are from 0 to 7.
UseExtRefClock	Yes, No	No	Use external reference clock. Applicable for PE*Trainer* ML only.
TrainerReset	Yes, No	No	When set, resets PE*Trainer* before script execution.

Example 1:

The following example configures PE*Trainer* to generate traffic on an x4 link (**LinkWidth = 4**) as a host emulator (**DirectionRx = Upstream**) and invert polarity on the first two lanes on incoming traffic (**InvertPolarityRx = (1,1,0,0)**).
The PE*Trainer* is reset before script execution (**TrainerReset = Yes**).
All options that are not specified (**DisableScrambleTx**, **DisableDescrambleRx**, **ReverseLanesTx**, **ReverseLanesRx**, **InvertPolarityTx**, **BaseSpec10**, **SkewTx**, and **UseExtRefClock**) are taken from the Generation Options dialog.

```
Config = General {
      LinkWidth = 4
      DirectionRx = Upstream
      InvertPolarityRx = (1,1,0,0)
      TrainerReset = Yes
}
```

Example 2:

The following example configures PE*Trainer* to generate traffic on an x8 link (**LinkWidth = 8**) as a device emulator (**DirectionRx = Downstream**).
Outgoing lanes are reversed (**ReverseLanesTx = Yes**).
Polarity on the last four outgoing lanes on outgoing traffic would be inverted (**InvertPolarityTx = (0,0,0,0,1,1,1,1)**).
Lanes 0 and 4 would have a skew value of 1 symbol time.
PE*Trainer* is reset before script execution (**TrainerReset = Yes**).

```
Config = General
{
      LinkWidth = 8
      DirectionRx = Downstream
      SkewTx = (1,0,0,0,1,0,0,0)
      InvertPolarityTx = ( 0,0,0,0,1,1,1,1 )
      ReverseLanesTx = Yes
      TrainerReset = Yes
}
```

7.2 Config = FCTx

This command allows the user to specify the policy for TLP transmission in regards to received Flow Control DLLP packets.

Parameter	Values	Default	Comment
CareForFC	Yes, No	Yes	When not set, the TLP packets are sent without regard for how many credits are available.

Example:

In this example, Flow Control checking is turned off for outgoing TLP packets.
The TLP packets that are declared after this **Config = FCTx** command are sent without checking for available FC credits.

```
Config = FCTx {
      CareForFC = No
}

Packet = TLP {
      TLPType = CfgRd0
      Length = 1
      Register = 0
      Count = 10000
}
```

7.3 Config = FCRx

This command configures automatic **UpdateFC** DLLP generation.

Parameter	Values	Default	Comment
Timer	In ns (rounded to nearest 8), Off	4200	Periodic timer that controls the sending of **UpdateFC** DLLP packets
PH	0:255	1	Posted Request Headers
NPH	0:255	1	Non-Posted Request Headers
CplH	0:255	1	Completion Headers
PD	0:4095	1024	Posted Request Data Payload
NPD	0:4095	1	Non-Posted Request Data Payload
CplD	0:4095	1024	Completion Data Payload

Example:

In this example, the timer for sending Update FC DLLP packets is specified.
Also, the initial number of FC credits for headers to advertise is specified.
The default values would be used for data credits.

```
Config = FCRx {
      Timer = 4000    ; Send UpdateFC DLLP packets every 4000 ns
      PH = 1          ; 1 credit for Posted Request Headers
      NPH = 2         ; 2 credits for Non-Posted Request Headers
      CplH = 0        ; Infinite number of credits for Completion
Headers
}
```

7.4 Config = TLP

This command facilitates data integrity control.

Parameter	Values	Default	Comment
AutoSeqNumber	Yes, No	Yes	If set to 0, overrides automatic generation of the TLP sequence number and uses the user-defined value as set in the **Packet = TLP** command (see Page 4).
AutoLCRC	Yes, No	Yes	If set to 0, overrides automatic generation of the TLP LCRC and uses the user-defined value as set in the **Packet = TLP** command (see Page 4).
ReplayTimer	In ns (rounded to nearest 8), Off	4200	Timeout in the TLP transmitter path that counts time since the latest **Ack** or **Nak** DLLP was received. If set, automatically retransmits TLP packets that were **Nak'**ed or on replay timer expiration.
AutoRetrain	Yes, No	Yes	If set, enables automatic retraining of the link in case the number of retransmitted TLPs is 4. Applicable only when the **ReplayTimer** is not turned off.
TagGeneration	Manual, Default, Extended, Phantom1, Phantom2, Phantom3	Manual	Tag generation policy for posted TLP packets: **Manual**: Tags are taken from the script. **Default**: Use lower 5-bits of Tag field. Zero out higher 3 bits. **Extended**: Use 8-bits of Tag field. **Phantom1**: Use 1 most significant bit of the Function field and 8-bits of Tag. **Phantom2**: Use 2 most significant bits of the Function field and 8-bits of Tag. **Phantom3**: Use 3 bits of Function field and 8-bits of Tag.

Example:

This example shows how to turn off automatic PSN and LCRC generation for outgoing TLP packets. The **ReplayTimer**, **AutoRetrain**, and **TagGeneration** parameters are omitted so the default values are used.

```
Config = TLP {
      AutoSeqNumber = No
      AutoLCRC = No
}
```

7.5 Config = AckNak

Parameter	Values	Default	Comment
AckNak	Auto, Ack, Nak, Disable	Auto	**Auto**: Automatic Ack/Nak **Ack**: Always Ack **Nak**: Always Nak **Disable**: Disable automatic Ack/Nak DLLP generation.
Delay	In ns (rounded to nearest 8)	0	Timer that controls how much delay is added to Ack/Nak DLLP response after TLP reception. Valid if **AckNak** is set to **Auto**, **Ack**, or **Nak.**

Example:

This example shows how to configure the PE*Trainer* so it **Naks** each incoming TLP packet.

```
Config = AckNak {
      AckNak = Nak
}
```

7.6 Config = Transactions

This command determines the behavior of PE*Trainer* as it responds to Memory, Configuration, and IO TLP requests. So that it properly responds to Memory and IO TLP requests, Configuration Address Space must be defined (see Page 62)

Parameter	Values	Default	Comment
AutoCfgCompletion	Yes, No	No	If set, automatically handles Configuration Read and Write TLP transactions. For a Configuration Read transaction, Completion TLP contains the data read from the internal Configuration Space according to specified register address. For a Configuration Write transaction, internal Configuration Space is updated at the address with the data taken from Configuration Write TLP, and a Configuration Write Completion is returned.
AutoMemIoCompletion	Yes, No	No	If set, automatically handles Memory and IO Read/Write TLP transactions. For Memory and IO Read transactions, a Completion TLP contains the data read from the internal Memory/IO Address Space according to specified address. For Memory and IO Write transactions, internal Memory/IO Address Space is updated at the address with the data taken from the TLP. (PE*Trainer* EML only)
EnableUR	Yes, No	No	If set, enables Unsupported Request (UR) status for Memory/IO completions. **AutoMemIoCompletion** must be set to enable UR completions. (PE*Trainer* EML only)
EnableCA	Yes, No	No	If set, enables Completer Abort (CA) status for Memory/IO completions. **AutoMemIoCompletion** must be set to enable CA completions. (PE*Trainer* EML only)
Poisoned	Yes, No	No	If set, all Memory/IO completions have the **Poisoned** bit set. (PE*Trainer* EML only)

Example:
This example enables automatic completion for Configuration TLP requests.
To automatically complete Configuration TLP requests, the Configuration Space must be configured first (see Page 64).

```
Config = Transactions {
      AutoCfgCompletion = Yes
      ; Automatically complete Configuration TLP requests.
}
```

Note: After this command, automatic completion for Memory and I/O TLP requests are turned off, since the default value (**No**) is used for the **AutoMemIoCompletion** parameter.

7.7 Config = Link

Parameter	Values	Default	Comment
FTSCount	0:255	255	Number of FTS ordered sets required (as sent in TS)
ExtendedSynch	Yes, No	Yes	When set, forces the transmission of 4096 FTS ordered sets.
SkipTimer	In ns (rounded to nearest 8), Off	4720	Periodic timer that controls sending of SKIP ordered sets at specific intervals. Timer's value is measured in 1us units.

Example:

This example configures the number of Fast Training Sequences to send when transitioning from L0s state (see Page 30).
This number also is advertised during Link Training.
This command also configures the periodic timer for SKIP Ordered Sets – sent every 4700 ns.

```
Config = Link
{
      SkipTimer = 4700
      FTSCount = 255
}
```

7.8 Config = Definitions

Parameter	Values	Default	Comment
Any literal	Any integer, string, array or predefined value		The defined values can be used anywhere in the script as a parameter value.

Example 1:

```
Config = Definitions {
      my_register     = 0x24
      my_tlptype      = CfgWr0
      my_payload      = ( 0x12345678 )
      my_wait_message = “my wait”
}
Packet = TLP {
      PSN      = Incr
      TlpType  = my_tlptype
      Register = my_register
      Payload  = my_payload
}
Config = Definitions {
      my_register = 0x20
      my_tlptype  = CfgWr1
}
Packet = TLP {
      PSN      = Incr
      TlpType  = my_tlptype
      Register = my_register
      Payload  = my_payload
}
wait = my_wait_message
```

Example 2:

This example shows how to use definitions in the expressions (see Page 56) and how to redefine the values.

```
Config = Definitions {
      READ_START = 0x10
}
; Repeat 10 times.
Repeat = Begin {
      Count=10
      Counter = i
}
; Send TLP using repeat counter (i) and
; READ_START to specify the address.
Packet = TLP {
      TLPType = CfgRd0
      Register = ( READ_START + ( 4 << i ) )
}
Repeat=End
; Redefine READ_START, now READ_START is 0x40.
Config = Definitions
{
      READ_START = ( READ_START + 0x30 )
}
; Send TLP using READ_START to specify the address.
Packet = TLP {
      TLPType = CfgRd0
      Register = READ_START
}
```

8 Wait Command

This command yields script execution until condition specified is true or timeout expires.

Parameter	Values	Default	Comment
Timeout		0	Timeout in nanoseconds, **0** means infinite timeout.
Display	Any string literal		Message displayed during the waiting in status bar
Count	1: 65535	1	Repeats wait specified number of times.

8.1 Wait = TLP

This command waits for a TLP that matches the defined condition. Only TLP Header fields can be specified. All parameters from **Packet = TLP** command (see Page 4) are valid, except **PSN**, **ECRC**, **LCRC** and **Payload** parameters.

TLP Header fields can be masked using the following format:

`0x0XAXX` For hexadecimal values
`0b0001XX` For binary values

Example:

This command waits infinitely for a Configuration Write request to registers from 0x1000 to 0x1FFF.

```
Wait = TLP {
      TLPType = CfgWr
      Register = “0x1XXX”
      Timeout = 0
}
```

8.2 Wait = DLLP

This command waits for a DLLP that matches the defined condition. All parameters from **Packet = DLLP** command (see Page 17) are valid, except the **CRC** field.

DLLP fields can be masked using the following format:

`0x0XAXX`	For hexadecimal values
`0b0001XX`	For binary values

Example 1:

This command waits for Ack DLLP.
The execution continues when Ack DLLP is received or after the 256 ns timeout expires.

```
Wait = DLLP {
      DLLPType = Ack
      Timeout = 256
}
```

Example 2:

This command waits for a Vendor DLLP with the Least Significant Bit of the vendor specific data set.
The execution continues when such DLLP is received or after the 256 ns timeout expires.

```
Wait = DLLP {
      DLLPType = Vendor
      VendorSpecific = "0bXXXXXXXXXXXXXXXXXXXXXXX1"
      Timeout = 256
}
```

8.3 Wait = Error

Parameter	Values	Default	Comment
Errors	DLLPCRC TLPLCRC Delimiter Disparity Symbol IdleData SkipLate OrdSetFormat EndBadPacket		The list of errors to wait for. If not specified, this waits for any error.

Example:

This command waits for a **Delimiter**, **Disparity**, or **Symbol** error to occur in incoming traffic.
The script continues running when any of the specified errors occur or after the 1024 ns timeout expires.

```
Wait = Error {
      Errors = (Delimiter, Disparity, Symbol)
      Timeout = 1024
}
```

8.4 Wait = LinkCondition

Parameter	Values	Default	Comment
Conditions	SKIP IDLE TS1 TS2 FTS PATN DLLP TLP COMMA		The list of conditions to wait for.
TrainingControl	(X,X,X,X)		(PE*Trainer*™ EML only). Training control bits. The order of the bits is as follows: (**HotReset**, **DisableLink**, **Loopback**, **DisableScrambling**)

Example 1:

This command waits for the COMMA symbol in incoming traffic.
The script execution continues when the COMMA symbol is received or after the 1024 ns timeout expires.

```
Wait = LinkCondition {
      Conditions = ( COMMA )
      Timeout = 1024
}
```

Example 2:

This command waits for a Training Sequence Ordered Set (TS1 or TS2) in incoming traffic with the **HotReset** bit asserted in the **TrainingControl** bits.
The script execution also continues after the 1024 ns timeout expires.

```
Wait = LinkCondition {
      Conditions = ( TS1, TS2 )
      TrainingControl = ( 1, 0, 0, 0 )
      Timeout = 1024
}
```

8.5 Wait = BOB

This command waits for Breakout Board Data match.

Parameter	Values	Default	Comment
Data			Mask and Match four bits of Breakout Board data

Example:

This command waits for Breakout Board Data with the Least Significant Bit and Most Significant Bit set.

```
Wait = BOB {
      Data = “0b1XX1”
}
```

8.6 Wait = Payload

This command waits for TLP payload match.

Parameter	Values	Default	Comment
Data			Mask and Match up to four DWORDs of TLP payload PE*Trainer* ML: Any offset from the beginning of payload PE*Trainer* EML: Zero offset from the beginning of payload
Data@<offset>			Mask and Match up to four DWORDs of TLP payload starting from **<offset>** offset from the beginning of payload (PE*Trainer* EML only)

Up to four DWORDs of the payload can be specified.

Example 1:

This command waits for a TLP with data payload 0x12345678.

Note: When this command is executed on PE*Trainer* EML, it matches only the first DWORD of the TLP payload.
When this command is executed on PE*Trainer* ML, it matches any DWORD from the TLP payload.

Script execution continues when a TLP with the specified payload is received or after the 1024 ns timeout expires.

```
Wait = Payload {
      Data = ( 0x12345678 )
      Timeout = 1024
}
```

Example 2:

This command waits for a TLP with a data payload that matches the following criteria:
1) The 1st DWORD's upper-most word must have **0xABCD**.
2) The 4th DWORD's lowest word must have **0x1234**.
3) The 2nd and 3rd DWORDs are insignificant.

Only the first four DWORDs of a TLP payload are checked when this command is executed on PE*Trainer* EML.

Any four subsequent DWORDs of a TLP payload are checked when this command is executed on PE*Trainer* ML.

Script execution continues when a TLP with specified payload is received or after the 1024 ns timeout expires.

```
Wait = Payload {
      Data = ( 0xABCDXXXX, 0xXXXXXXXX, 0xXXXXXXXX, 0xXXXX1234 )
      Timeout = 1024
}
```

Example 3:

The following example can be executed only on PE*Trainer* EML.

This command waits for a TLP with a data payload that matches the following criteria:

1) The 3rd DWORD's upper-most word must have **0xABCD**.
2) The 9th DWORD's lowest word must have **0x1234**.
3) The 10th DWORD's upper-most byte must have 0x56.

Script execution continues when a TLP with specified payload is received or after the 1024 ns timeout expires.

```
Wait = Payload {
      Data@2 = ( 0xABCDXXXX )
      Data@8 = ( 0xXXXX1234, 0x56XXXXXX )
      Timeout = 1024
}
```

8.7 Wait = User

This command waits for user input. The script execution would continue when user resumes the script from PE*Tracer*™ software UI.

Example:

This example would pause the script execution and display the message to the user.

```
Wait = User {
      Display = “Now you can continue”
}
```

8.8 Additional “Wait” Modifiers

1. **Wait = <number>**

   Unconditionally yields script execution for the specified number of nanoseconds.

   Example:

   ```
   Wait = 500
   ```

2. **Wait = <Text>**

   Equivalent to

   ```
   Wait = User {
         Display = <Text>
   }
   ```

   Example:

   ```
   Wait = “Press the button to continue script execution”
   ```

   A count parameter can be applied to this command, which causes it to wait for that number clicks on the user input button.

9 Branch Command

This command enables/disables interrupt for the condition specified.

9.1 Branch = <condition>

This command enables the interrupt for the condition specified. The conditions are the same as in the **Wait** command (see Page 28), except **User**. The parameter list is the same as the **Wait** command, except for the **Timeout**, **Display**, and **Count** parameters.

Here is a list of additional parameters for the **Branch = <Condition>** command:

Parameter	Values	Default	Comment
BranchName	Any string literal		Name of the branch. Must be specified if this branch is to be disabled later.
ProcName	Any string literal		Name of the procedure to execute when branch conditions are met.

ProcName parameter is mandatory.
BranchName parameter could be omitted if you do not plan to disable the branch later in the script.

The procedure that handles the branch condition must be defined before the **Branch = <Condition>** command (see Page 35).

Example:

```
…
Proc = Begin {
      ProcName = “Procedure1”
}
...

Proc = End

; The following statement specifies that if Delimiter, Disparity
; or Symbol error occurs, then the code declared in “Procedure1”
; should be executed.

Branch = Error {
      BranchName = “SomeErrorBranch”
      ProcName = “Procedure1”
      Errors = (Delimiter, Disparity, Symbol)
}
...

; Disable the branch “SomeErrorBranch” that is specified above.

Branch = Disable {
      BranchName = “SomeErrorBranch”
}
...
```

9.2 Branch = Disable

This command disables the interrupt that was previously enabled.

Parameter	Values	Default	Comment
BranchName	Any string literal		Name of the branch

Branch with the name specified in **BranchName** parameter must be defined.

10 Proc Command

This command declares the procedure to be executed for the **Branch** command. Procedure declaration must precede its usage in the **Branch** statement.

10.1 Proc = Begin

This command declares the start point of the procedure.

Parameter	Values	Default	Comment
ProcName	Any string literal		Name of the procedure

10.2 Proc = End

This command declares the end point of the procedure.

11 Loop Command

This command causes the PE*Trainer*™ BusEngine™ to re-execute a block of commands a predefined number of times.

Note: Loops require up to 1 us to branch to the beginning of the loop. During this time, script execution is paused. Internally generated packets, such as SKIP ordered sets, Ack/Nak DLLP packets, and flow control updates, still occur as programmed.

Loops can be nested up to 4 deep.

11.1 Loop = Begin

This command marks the beginning of the loop.

Parameter	Values	Default	Comment
Count	0:65535 Infinite		Specifies how many times to repeat the loop. Setting Count to 0 causes an infinite loop.

11.2 Loop = End

This command marks the end of the loop.

Example:

```
Loop = Begin { count = 10 }
      Packet = TLP { TLPType = CfgRd0  Length = 1  Register = 0 }
Loop = End
```

12 Repeat Command

This command causes one or more commands to be repeated. This is not implemented as a branch instruction in the BusEngine™, but is a replication of commands during script compilation in the software.

This allows back-to-back execution of these commands with as little as 0 symbol times of IDLE traffic between them.

This command increases the size of the script object that is downloaded to the PE*Trainer*™ and increases download time accordingly.

12.1 Repeat = Begin

This command marks the beginning of the code being repeated.

Parameter	Values	Default	Comment
Count	1:65535		Values of Infinite and 0 are not supported
Counter			

12.1.1 Counter Parameter

Any string literal can be used for the **Counter** parameter.

The value of the **Counter** parameter can be used within the **Repeat** statement (i.e., between **Repeat=Begin** and **Repeat=End**) in arithmetic expressions for any parameter, except the parameters that require the array data type (such as Payload for TLP packet).

The value of the **Counter** parameter changes from 0 to the value of the **Count** parameter minus one.

Arithmetic expressions must be included in round brackets (parentheses).

The operators are: +, -, *, /, <<, >>, &, |, ~.

Example 1:

Within this repeat, **ppp** can be used in arithmetic expressions for any packet field. The value of **ppp** changes from 0 to 3 in the example.
The **Tag** parameter accepts the values **0x10, 0x11, 0x12**, and **0x13**.
The **AddressHi** parameter accepts the values **0x00400000, 0x00400001, 0x00400001**, and **0x00400002.**

```
Repeat = Begin { Count = 4 Counter = ppp }

Packet = TLP {
            TLPType = MRd64
            Tag = ( ppp + 0x10 )
            AddressHi = ( 0x400000 + 4 / ( 5 - ppp ) )
}

Repeat = End
```

Example 2:

The following example shows the usage of the counters in nested repeats. The counter **qqq** is used for the outer repeat. The counter **www** is used for the inner repeat. **Packet = TLP** in the inner repeat uses both counters to construct the **AddressHi** parameter.

```
Repeat = Begin { Count = 3 Counter = qqq }

Packet = DLLP {
      DLLPType = Ack
      AckNak_SeqNum = ( qqq + 1 )
}

Packet = DLLP {
      DLLPType = Ack
      AckNak_SeqNum = ( 0xf & ~qqq )
}

      Repeat = Begin { Count=4 Counter = www }

      Packet = TLP {
            TLPType = MRd64
            AddressHi = ( 0x400000 + www * 4 + qqq ) )
      }

      Repeat = End

Repeat = End
```

12.2 Repeat = End

This command marks the end of the code being repeated.

Example:

```
Repeat = Begin { count = 10 }
      Packet = TLP { TLPType = CfgRd0  length = 1 register = 0 }
Repeat = End
```

13 Template Command

This command creates a template for a packet that can be used in the **Packet** command. The fields specified in the **Template** command may be overridden in the **Packet** command.

Example 1:

The following example issues three Memory Read requests.

```
Template = TLP {
      Name = “TestPacket”
      Type = MRd32
      TC = 0
      Tag = 0
      RequesterID = (1:0:0)
      Length = 64
      Address = 0
}

Packet = “TestPacket” {
}

Packet = “TestPacket” {
      Address = 64
}

Packet = “TestPacket” {
      Address = 128
}
```

Example 2:

The following example shows nested templates (i.e. when one template is based on another template).

```
; First define the template "SomeTlp3" for TLP packet.

Template = TLP {
      Name = "SomeTlp3"
      TLPType = MRd32
      RequesterID = (0:1:2)
      Length = 0x40
      LastDwBe = 0xF
      FirstDwBe = 0xF
      Address = 0x10000
}

; The template "SomeTlp4" is based on the template "SomeTlp3"
; with Address overridden.

Template = "SomeTlp3" {
      Name = "SomeTlp4"
      Address = 0x10040
}

; This TLP packet has Address parameter equal to 0x10000.

Packet = "SomeTlp3" {
      Length = 0x80
}

; This TLP packet has Address parameter equal to 0x10040.

Packet = "SomeTlp4" {
      Length = 0x80
}
```

14 Include Command

This command includes the PE*Trainer*™ script file inline. All commands in the included file are executed, with the exception of the **Config = General** command.

The format of this command is following:

```
Include = <file_path>
```

where **file_path** is a path to the file to be included. If **file_path** is not a fully qualified path, then the relative path to the current script file would be used.

Example 1:

In this example, all commands from the **included1.peg** file would be executed first, then all commands from the **included2.peg** file would be executed, and then the 32-bit Memory Read TLP would be sent.

```
Include = "included1.peg"     ; All packets from included1.peg file
                              ; would be inserted here.
Include = "included2.peg"     ; All packets from included2.peg file
                              ; would be inserted here.

Packet = TLP           ; Sending 32-bit Memory Read TLP request
{
      TLPType = MRd32  ;  Memory Read request (32 bit)
      TC = 0x7         ; Traffic class is 7.
      TD = 0x1         ; TLP digest is present.
      EP = 0x0         ; TLP is not poisoned.
      Address = 0x1000 ; Reading from address 1000h of memory space
      Length = 0x40    ; Reading 40h DWORDs
}
```

Example 2:

The first command of this example includes all commands from the file **c:/Testing/included1.peg**.
If we assume that the current script is located in the folder **c:/Testing/TLP**, then the second command of this example includes all commands from the file **c:/Testing/TLP/included2.peg**.
If we assume that the current script is located in the folder **c:/Testing/TLP**, then the third command of this example includes all commands from the file **c:/Testing/included3.peg**.

```
Include = "c:/Testing/included1.peg"; All packets from included1.peg
                                    ; file would be inserted here.
Include = "included2.peg"           ; All packets from included2.peg
                                    ; file would be inserted here.
Include = "../included3.peg"        ; All packets from included3.peg
                                    ; file would be inserted here.
```

15 AddressSpace Command

This command reads/writes the PE*Trainer*™ memory region.

PE*Trainer* maps Memory and IO address spaces to its internal memory region according to Base Address Registers (BAR) specified in the Configuration Address Space.

PE*Trainer* uses its memory regions when processing Memory, IO, and Configuration TLP requests (see Section 0).

PE*Trainer* maps Configuration address space to its internal memory region (**Cfg**).

PE*Trainer* supports one 64-bit Memory region, two 32-bit Memory regions, and two IO Memory regions.

Maximum address space sizes supported by PE*Trainer* are as follows:

Address Space	**Size**
Configuration	4 KB
32-bit memory	128 MB
64-bit memory	512 MB
IO	256 MB

Mapping of BARs to PE*Trainer* memory regions:

Memory Region	**BAR**
Mem64	First BAR that defines 64-bit Memory Address Space
Mem32A	First BAR that defines 32-bit Memory Address Space
Mem32B	Second BAR that defines 64-bit Memory Address Space
IOA	First BAR that defines IO Address Space
IOB	Second BAR that defines IO Address Space

In order to properly respond to Memory and IO TLP requests, the Configuration space must be written to the PE*Trainer* first.

Mem64, Mem32A, Mem32B, IOA, and IOB memory regions are not implemented in PE*Trainer* ML.

15.1 AddressSpace = Read

This command reads specified memory region from PE*Trainer* and stores it in specified file.

Parameter	Values	Default	Comment
Location	Cfg Mem64 Mem32A Mem32B IOA IOB		Specifies the memory region to read from. The memory region is mapped to address space according to the rules described above. **Mem64**, **Mem32A**, **Mem32B**, **IOA**, and **IOB** are applicable to PE*Trainer* EML only.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies offset in bytes from the beginning of memory region specified in the **Location** parameter.
Size	Any number from 0. The combination of **Offset** and **Size** parameters is limited by the maximum allowed address. (The maximum allowed address is determined by memory region specified in the **Location** parameter.)	Maximum allowed size for memory region specified in the **Location** parameter	Specifies number of bytes to read starting from the address specified in the **Offset** parameter.
SaveTo	Any file path		File path to store the memory read.

Example 1:

This command reads the whole Mem32A memory region and stores it in the **c:/mem.bin** file.
The offset is 0. The read size is 128MB.

```
AddressSpace = Read {
      Location = Mem32A
      SaveTo = "c:/mem.bin"
}
```

Example 2:

This command reads 16 bytes from address 0x1000 of **Mem64** memory region and stores it in the **c:/mem.bin** file.

```
AddressSpace = Read {
      Location = Mem64
      Offset = 0x1000
      Size = 0x10
      SaveTo = "c:/mem.bin"
}
```

15.2 AddressSpace = Write

This command writes specified memory region into PE*Trainer* from specified data source.

During write operations into Mem64, Mem32A, Mem32B, IOA, and IOB regions, the automatic completions of Memory and IO TLP requests are disabled (see Section 0).

During write operations into Cfg region, the automatic completions of Configuration TLP requests are disabled (see Section 0).

Parameter	Values	Default	Comment
Location	Cfg Mem64 Mem32A Mem32B IOA IOB		Specifies the memory region to write into. The memory region is mapped to address space according to the rules described above. **Mem64**, **Mem32A**, **Mem32B**, **IOA**, and **IOB** are applicable to PE*Trainer* EML only.
Offset	Any number from 0 to the maximum allowed address determined by the memory region specified in the **Location** parameter	0	Specifies offset in bytes from the beginning of the memory region specified in the **Location** parameter.
Size	Any number from 0. The combination of **Offset** and **Size** parameters is limited by the maximum allowed address. (The maximum allowed address is determined by memory region specified in the **Location** parameter.)	If **Zeros**, **Ones**, **Random**, or **Incr** is specified for the **LoadFrom** parameter, then the default value is the maximum allowed size for the memory region specified in the **Location** parameter. Otherwise, the default size is the size of data specified in the **LoadFrom** parameter.	Specifies number of bytes to write starting from the address specified in the **Offset** parameter.
LoadFrom	Any file path Any array of bytes Zeros Ones Random Incr	Zeros	

Example 1:

This command clears the whole **Mem32A** memory region.

```
AddressSpace = Write {
      Location = Mem32B
      LoadFrom = Zeros
}
```

Example 2:

This command writes 16 bytes, starting from address **0x1000**, into the **Mem64** memory region from file **c:/mem.bin**.

```
AddressSpace = Write {
      Location = Mem64
      Offset = 0x1000
      Size = 0x10
      LoadFrom = "c:/mem.bin"
}
```

Example 3:

This command writes 7 bytes, starting from address **0x1000**, into the **Mem64** memory region from data specified.

```
AddressSpace = Write {
      Location = Mem64
      Offset = 0x1000
      LoadFrom = ( 0x02, 0x08, 0x01, 0x03, 0x06, 0x07, 0x07 )
}
```

Example 4:

This command writes 48 bytes of random data, starting from address **0x10**, into the **IOA** memory region.

```
AddressSpace = Write {
      Location = IOA
      Offset = 0x10
      Size = 0x30
      LoadFrom = Random
}
```

How to Contact LeCroy

Type of Service	**Contact**
Call for technical support…	US and Canada: 1 (800) 909-2282 Worldwide: 1 408-653-1260
Fax your questions…	Worldwide: 1 (408) 727-6622
Write a letter …	LeCroy Protocol Solutions Group Customer Support 3385 Scott Blvd. Santa Clara, CA 95054
Send e-mail…	support@catc.com
Visit LeCroy's web site…	http://www.lecroy.com/