



Protocol Solutions Group

# Scripting and Automation API

## Reference Manual

for

# Summit M5x Analyzer/Jammer

For PCIe Protocol Analysis version 11.37

Generated: 3/10/2020 9:23 AM

## Document Disclaimer

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

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

## Trademarks and Servicemarks

*Teledyne LeCroy, CATC Trace, PETracer, PCIe Protocol Analysis, PCIe Protocol Suite, Summit, Summit T3-16, Summit T3-8, Summit T34, Summit T28, Summit T24, Summit Z3-16, Summit Z416, Summit T416, Summit T48, Summit M5x and BusEngine* are trademarks of Teledyne LeCroy.

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

All other trademarks are property of their respective companies.

## Copyright

© 2012 Teledyne LeCroy, Inc. All Rights Reserved.

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

## Table of Contents

<b>1</b>	<b>Introduction</b> .....	<b>1</b>
1.1	System Requirements .....	1
1.2	Support Resources .....	1
1.3	Setting Up Automation for Local Use .....	1
1.4	Setting Up Automation for Remote Use .....	1
<b>2</b>	<b>Script Mode Overview</b> .....	<b>2</b>
2.1	PCIe Protocol Analysis Software .....	2
2.1.1	Recording Options .....	3
2.1.2	Recording Options: Recording Rules .....	4
2.1.3	Recording Rules: New Events .....	5
2.2	Converting Recording Rules and Jammer Actions from GUI to Script File .....	6
2.2.1	Complex Recording Rules (1) .....	6
2.2.2	Complex Recording Rules (2) .....	7
2.2.3	Jammer Options .....	8
2.3	Script Mode .....	9
2.3.1	Script Parsing .....	12
2.4	Commands .....	13
2.5	Events .....	14
2.6	Jammer Actions .....	15
<b>3</b>	<b>Scripting Jammer Functionality</b> .....	<b>16</b>
3.1	Example: Select Jammer Action: Delete .....	16
3.2	Recording Options and Jammer Action Script .....	21
3.3	Updating Jammer Action Script .....	23
3.4	Warning Message if Mistake in Jammer Script .....	28
3.5	Updated GUI After Scripting Mistake Corrected .....	30
3.6	Saving a Jammer Actions Script File .....	31
3.7	Loading a Jammer Script File .....	32
3.8	Examples of Scripting for Events with Jammer Actions 37	
3.8.1	Link State: Speed Change .....	37
3.8.2	Ordered Set: TS2 .....	41
3.8.3	Framing Token: EDS .....	43
3.8.4	Errors: Symbol Error .....	45
3.8.5	DLLP: Any .....	47
3.8.6	TLP: Any .....	49
3.8.7	NVMe Events .....	51

<b>4</b>	<b>Detailed Description of Script Mode Syntax .....</b>	<b>57</b>
4.1	General Syntax Rules.....	57
4.2	Command List .....	58
4.3	RecEvent Command.....	59
4.3.1	Common for all RecEvent fields .....	59
4.3.2	RecEvent = TLP.....	61
4.3.3	RecEvent = DLLP .....	74
4.3.4	RecEvent = OrderedSet.....	83
4.3.5	RecEvent = TS1, TS2 .....	84
4.3.6	RecEvent = Skip .....	88
4.3.7	RecEvent = CtrlSkip.....	89
4.3.8	RecEvent = Lane_Margining.....	90
4.3.9	RecEvent = LinkState .....	92
4.3.10	RecEvent = Error.....	94
4.3.11	RecEvent = NVMe_Register.....	95
4.3.12	RecEvent = NVMe_Command_Submission .....	97
4.3.13	RecEvent = NVMe_Command_Completion.....	99
4.3.14	RecEvent = MCTP.....	101
4.3.15	RecEvent = Global_Timer_A, Global_Timer_B, Local_Timer_A, Local_Timer_B, Local_Timer_C, Local_Timer_D .....	113
4.3.16	RecEvent = Global_Counter_1, Global_Counter_2, Local_Counter_1, Local_Counter_2, Local_Counter_3, Local_Counter_4 .....	114
4.4	RecAction Command .....	115
4.4.1	RecAction=Jammer.....	115
<b>5</b>	<b>PETracer Object Model .....</b>	<b>137</b>
<b>6</b>	<b>PEAnalyzer Object.....</b>	<b>140</b>
6.1	IAnalyzer interface .....	141
6.1.1	IAnalyzer::GetVersion .....	142
6.1.2	IAnalyzer::OpenFile .....	143
6.1.3	IAnalyzer::StartGeneration .....	144
6.1.4	IAnalyzer::StopGeneration .....	145
6.1.5	IAnalyzer::StartRecording .....	146
6.1.6	IAnalyzer::StopRecording.....	148
6.1.7	IAnalyzer::MakeRecording.....	149
6.1.8	IAnalyzer::LoadDisplayOptions .....	150
6.1.9	IAnalyzer::GetRecordingOptions .....	151
6.1.10	IAnalyzer::GetSerialNumber .....	152
6.2	IPEAnalyzer interface.....	153
6.2.1	IPEAnalyzer::GetGenerationOptions .....	154
6.2.2	IPEAnalyzer::ResumeGeneration .....	155
6.2.3	IPEAnalyzer::GetLinkStatus .....	156

6.2.4	IPEAnalyzer::GetLinkConfigEx .....	157
<b>6.3</b>	<b>IPEAnalyzer2 interface .....</b>	<b>158</b>
6.3.1	IPEAnalyzer2::GetHardwareInfo .....	159
6.3.2	IPEAnalyzer2::ResetHardware .....	161
<b>6.4</b>	<b>IPEAnalyzer3 interface .....</b>	<b>162</b>
6.4.1	IPEAnalyzer3::StartImportFile .....	163
<b>6.5</b>	<b>IPEAnalyzer5 interface .....</b>	<b>164</b>
6.5.1	IPEAnalyzer5::GetGenerationOptionsAsXML .....	165
6.5.2	IPEAnalyzer5::SetGenerationOptionsFromXML .....	166
<b>6.6</b>	<b>IPEAnalyzer6 interface .....</b>	<b>167</b>
6.6.1	IPEAnalyzer6::GetRecordingOptionsAsXML .....	168
6.6.2	IPEAnalyzer6::SetRecordingOptionsFromXML .....	169
<b>6.7</b>	<b>IPEAnalyzer8 interface .....</b>	<b>170</b>
6.7.1	IPEAnalyzer8::GetSerialNumberEx .....	171
<b>6.8</b>	<b>IPEAnalyzer23::SetDeviceRecordingEnabled .....</b>	<b>172</b>
<b>6.9</b>	<b>IPEAnalyzer28 interface .....</b>	<b>173</b>
6.9.1	IPEAnalyzer28::StartJammingAndRecording .....	174
6.9.2	IPEAnalyzer28::StopJammingAndRecording .....	175
6.9.3	IPEAnalyzer28::StartJamming .....	176
6.9.4	IPEAnalyzer28::StopJamming .....	177
<b>6.10</b>	<b>IPEAnalyzer31 interface .....</b>	<b>178</b>
6.10.1	IPEAnalyzer31::SwitchM5XRecordingType .....	179
<b>PETrace Object .....</b>		<b>180</b>
<b>6.11</b>	<b>ITrace interface .....</b>	<b>181</b>
6.11.1	ITrace::GetName .....	182
6.11.2	ITrace::ApplyDisplayOptions .....	183
6.11.3	ITrace::Save .....	184
6.11.4	ITrace::ExportToText .....	185
6.11.5	ITrace::Close .....	188
6.11.6	ITrace::ReportFileInfo .....	189
6.11.7	ITrace::ReportErrorSummary .....	190
6.11.8	ITrace::ReportTrafficSummary .....	193
6.11.9	ITrace::GetPacket .....	194
6.11.10	ITrace::GetPacketsCount .....	197
6.11.11	ITrace::GetTriggerPacketNum .....	198
6.11.12	ITrace::AnalyzerErrors .....	199
<b>6.12</b>	<b>IPETrace interface .....</b>	<b>201</b>
6.12.1	IPETrace::GetBusPacket .....	201
<b>6.13</b>	<b>IPETrace2 interface .....</b>	<b>202</b>
6.13.1	IPETrace2::IsLevelDecodeFinished .....	202
6.13.2	IPETrace2::SetTag .....	203
6.13.3	IPETrace2::GetTag .....	203

<b>6.14</b>	<b>_ITraceEvents interface</b>	<b>204</b>
6.14.1	ITraceEvents::OnLevelDecodeFinished	204
<b>6.15</b>	<b>IPETrace3 Interface</b>	<b>205</b>
6.15.1	IPETrace3::GetPacketEx	205
6.15.2	IPETrace3::GetPacketsCountEx	205
6.15.3	IPETrace3::GetTriggerPacketNumEx	206
6.15.4	IPETrace3::GetBusPacketEx	206
<b>6.16</b>	<b>IPEVerificationScript interface</b>	<b>207</b>
6.16.1	IPEVerificationScript::RunVerificationScript	208
6.16.2	IPEVerificationScript::GetVScriptEngine	210
<b>6.17</b>	<b>IPETraceForScript2 interface</b>	<b>212</b>
6.17.1	IPETraceForScript2::ImportXMLConfig	213
6.17.2	IPETraceForScript2::ImportXMLConfigFromFile	214
6.17.3	IPETraceForScript2::ExportXMLConfig	215
6.17.4	IPETraceForScript2::ExportXMLConfigToFile	216
6.17.5	IPETraceForScript2::Redecode	217
<b>6.18</b>	<b>IPETraceForScript3 interface</b>	<b>218</b>
6.18.1	IPETraceForScript3::ExportToText2	219
<b>6.19</b>	<b>IPETraceForScript4 interface</b>	<b>220</b>
6.19.1	IPETraceForScript4::GetTrafficSummaryAsXML	221
<b>6.20</b>	<b>IPETraceForScript5 interface</b>	<b>222</b>
6.20.1	IPETraceForScript5::ExportLinkTrackerToCsv	222
<b>6.21</b>	<b>IPETraceForScript6 interface</b>	<b>223</b>
6.21.1	IPETraceForScript5::SetOldExpFormat	224
<b>7</b>	<b>PERecOptions Object</b>	<b>225</b>
<b>7.1</b>	<b>IRecOptions interface</b>	<b>226</b>
7.1.1	IRecOptions::Load	226
7.1.2	IRecOptions::Save	227
7.1.3	IRecOptions::SetRecMode	228
7.1.4	IRecOptions::SetBufferSize	229
7.1.5	IRecOptions::SetBufferSizeEx	230
7.1.6	IRecOptions::SetPostTriggerPercentage	231
7.1.7	IRecOptions::SetTriggerBeep	232
7.1.8	IRecOptions::SetSaveExternalSignals	233
7.1.9	IRecOptions::SetTraceFileName	234
7.1.10	IRecOptions::Reset	235
<b>7.2</b>	<b>IPERecOptions interface</b>	<b>236</b>
<b>7.3</b>	<b>IPERecOptions2 interface</b>	<b>236</b>
7.3.1	IPERecOptions2::SetTargetAnalyzer	237
7.3.2	IPERecOptions2::SetLinkWidth	238
7.3.3	IPERecOptions2::SetBase10Spec	239
7.3.4	IPERecOptions2::SetExternalRefClock	240
7.3.5	IPERecOptions2::SetDisableDescrambling	241

7.3.6	IPERecOptions2::SetDisableDeskew .....	242
7.3.7	IPERecOptions2::SetAutoConfigPolarity .....	243
7.3.8	IPERecOptions2::SetInhibit .....	244
7.3.9	IPERecOptions2::SetReverseLanes .....	245
7.3.10	IPERecOptions2::SetInvertPolarity .....	246
<b>7.4</b>	<b>IPERecOptions3 interface .....</b>	<b>247</b>
7.4.1	IPERecOptions3::SetLinkSpeed .....	247
7.4.2	IPERecOptions3::GetSimpleTrigger .....	248
7.4.3	IPERecOptions3::GetSimpleFilter .....	250
7.4.4	IPERecOptions3::RestoreDefaultFactorySettings .....	251
<b>7.5</b>	<b>SimpleTrigger .....</b>	<b>252</b>
7.5.1	SimpleTrigger::SetEnabled .....	252
7.5.2	SimpleTrigger::IsEnabled .....	252
	Retrieves trigger's state .....	252
7.5.3	SimpleTrigger::SetDirection .....	252
7.5.4	SimpleTrigger::GetDirection .....	253
<b>7.6</b>	<b>SimpleFilter .....</b>	<b>254</b>
7.6.1	SimpleFilter::SetEnabled .....	254
7.6.2	SimpleFilter::IsEnabled .....	254
	Retrieves filter's state .....	254
7.6.3	SimpleFilter::SetDirection .....	254
7.6.4	SimpleFilter::GetDirection .....	255
<b>7.7</b>	<b>Script Mode Automation API .....</b>	<b>256</b>
<b>8</b>	<b>PEGenOptions Object .....</b>	<b>258</b>
<b>8.1</b>	<b>IGenOptions interface .....</b>	<b>259</b>
8.1.1	IGenOptions::Load .....	260
8.1.2	IGenOptions::Save .....	261
8.1.3	IGenOptions::Reset .....	262
<b>8.2</b>	<b>IPEGenOptions interface .....</b>	<b>263</b>
<b>8.3</b>	<b>IPEGenOptions2 interface .....</b>	<b>263</b>
8.3.1	IPEGenOptions2::SetTargetGenerator .....	264
8.3.2	IPEGenOptions2::SetWorkAsRoot .....	265
8.3.3	IPEGenOptions2::SetLinkWidth .....	266
8.3.4	IPEGenOptions2::SetBase10Spec .....	267
8.3.5	IPEGenOptions2::SetExternalRefClock .....	268
8.3.6	IPEGenOptions2::SetDisableDescrambling .....	269
8.3.7	IPEGenOptions2::SetDisableScrambling .....	270
8.3.8	IPEGenOptions2::SetAutoConfig .....	271
8.3.9	IPEGenOptions2::SetReverseLanes .....	272
8.3.10	IPEGenOptions2::SetInvertPolarity .....	273
8.3.11	IPEGenOptions2::SetSkew .....	274
<b>9</b>	<b>PEPacket Object .....</b>	<b>275</b>
<b>9.1</b>	<b>IPacket interface .....</b>	<b>276</b>

9.1.1	IPacket::GetTimestamp .....	276
<b>9.2</b>	<b>IPEPacket interface .....</b>	<b>277</b>
9.2.1	IPEPacket::GetPacketData.....	278
9.2.2	IPEPacket::GetLinkWidth.....	281
9.2.3	IPEPacket::GetStartLane .....	282
9.2.4	IPEPacket::GetDirection .....	283
9.2.5	IPEPacket::GetErrors .....	284
<b>10</b>	<b>PETraceErrors Object.....</b>	<b>285</b>
<b>10.1</b>	<b>IAnalyzerErrors dispinterface .....</b>	<b>285</b>
10.1.1	IAnalyzerErrors::get_Item.....	286
10.1.2	IAnalyzerErrors::get_Count.....	287
<b>11</b>	<b>PEVScriptEngine Object.....</b>	<b>289</b>
<b>11.1</b>	<b>IVScriptEngine interface .....</b>	<b>290</b>
11.1.1	IVScriptEngine::VScriptName .....	291
11.1.2	IVScriptEngine::Tag.....	292
11.1.3	IVScriptEngine::RunVScript .....	293
11.1.4	IVScriptEngine::RunVScriptEx .....	294
11.1.5	IVScriptEngine::LaunchVScript.....	296
11.1.6	IVScriptEngine::Stop .....	297
11.1.7	IVScriptEngine::GetScriptVar.....	298
11.1.8	IVScriptEngine::SetScriptVar .....	300
<b>12</b>	<b>PEVScriptEngine Object Events .....</b>	<b>302</b>
<b>12.1</b>	<b>_IVScriptEngineEvents interface.....</b>	<b>302</b>
12.1.1	_IVScriptEngineEvents::OnVScriptReportUpdated	305
12.1.2	_IVScriptEngineEvents::OnVScriptFinished .....	306
12.1.3	_IVScriptEngineEvents::OnNotifyClient.....	307
<b>13</b>	<b>PEAnalyzer Object Events .....</b>	<b>309</b>
<b>13.1</b>	<b>_IAnalyzerEvents dispinterface .....</b>	<b>309</b>
13.1.1	_IAnalyzerEvents::OnTraceCreated.....	310
13.1.2	_IAnalyzerEvents::OnStatusReport .....	311
<b>14</b>	<b>CATCAnalyzerAdapter.....</b>	<b>314</b>
<b>14.1</b>	<b>IAnalyzerAdapter Interface .....</b>	<b>315</b>
14.1.1	IAnalyzerAdapter::CreateObject .....	315
14.1.2	IAnalyzerAdapter::Attach.....	317
14.1.3	IAnalyzerAdapter::Detach .....	318
14.1.4	IAnalyzerAdapter::IsValidObject.....	320
<b>15</b>	<b>Teledyne LeCroy Internal Interfaces .....</b>	<b>321</b>
<b>16</b>	<b>Appendix A How to Contact Teledyne LeCroy .....</b>	<b>322</b>

# 1 Introduction

Teledyne LeCroy's PCIe Protocol Analysis™ software provides a rich, functional COM/Automation API to the most important functionalities of the Teledyne LeCroy Protocol Analyzer and Teledyne LeCroy Exerciser. This makes it a great tool for implementation of automated programs for complicated testing, development, and debugging. The "dual" nature of the interfaces provided makes it easy to use the *PE Tracer* COM API in different IDEs (Integrated Development Environment) supporting the COM architecture.

A special support for typeless script languages, like VB and JavaScript, while overriding some restrictions imposed by script engines (remote access, dynamic object creation, and handling events), gives the opportunity to write client applications very quickly and easily. One does not require significant programming skills nor installing expensive and powerful programming language systems. All these features, along with the ability to set up all necessary DCOM permissions during the installation process, make the Teledyne LeCroy Analyzer an attractive tool in automating and speeding up many engineering processes.

## 1.1 System Requirements

The Automation API was introduced with the following release: *PE Tracer* software 4.10. This document covers the functionality available in the latest PCIe Protocol Analysis.

## 1.2 Support Resources

As new functionalities are added to the API, not all of them are supported by older versions of the software. For newer releases of PCIe Protocol Analysis, please refer to the Teledyne LeCroy web site: [www.teledynelecroy.com](http://www.teledynelecroy.com)

## 1.3 Setting Up Automation for Local Use

If you intend to run Automation on the Summit Analyzer or Exerciser Host Controller (i.e., the PC attached to the Summit Analyzer or Exerciser), you do not need to perform any special configuration. You can simply execute the scripts or programs you have created and they run the analyzer. In order to use the PCIe Protocol Suite COM API, the application should be registered as a COM server in a system registry. This is done during the installation process.

## 1.4 Setting Up Automation for Remote Use

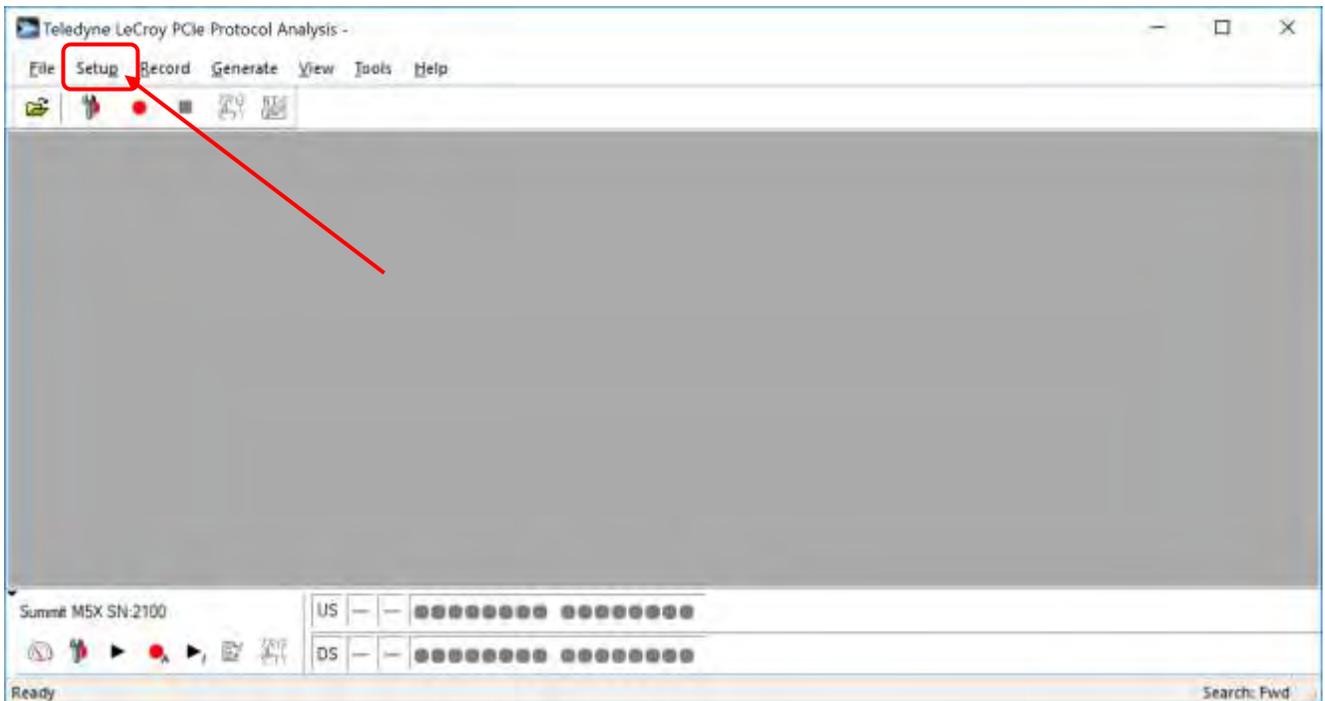
If you would like to access Summit Analyzer remotely over a network, you should install the PCIe Protocol Analysis application on both server and client machine and accept enabling remote access option during the installation. You can also perform a manual DCOM configuration.

## 2 Script Mode Overview

The Summit M5x can modify, replace, insert or delete real-time traffic between a PCIe root complex and endpoint to verify adherence to design specifications and identify potential errors in protocol behavior. The Summit M5x also introduces for the first time NVMe and NVMe-MI protocol jamming as well. The analyzer and jammer combination features on the Summit M5x make this a powerful testing tool for PCIe validation and development labs.

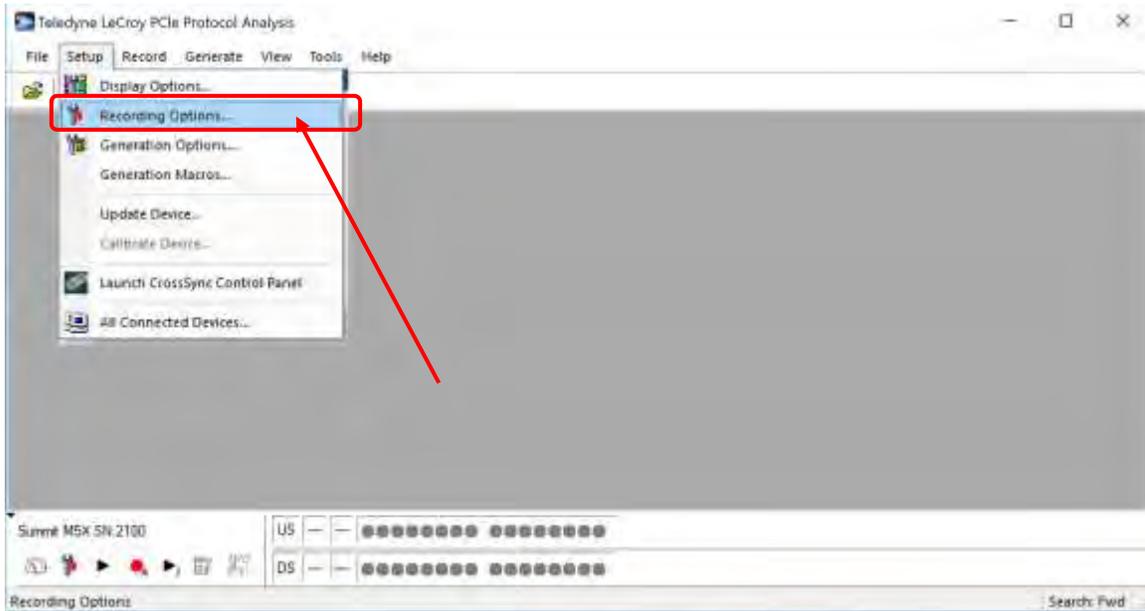
This Jammer and Scripting capability is built into the PCIe Protocol Analysis software. After the software has been loaded (as explained in more detail in the M5x Quick Start Guide), you can access the Jammer functionality through the Recording Options of the PCIe Protocol Analysis software dialog (see below).

### 2.1 PCIe Protocol Analysis Software

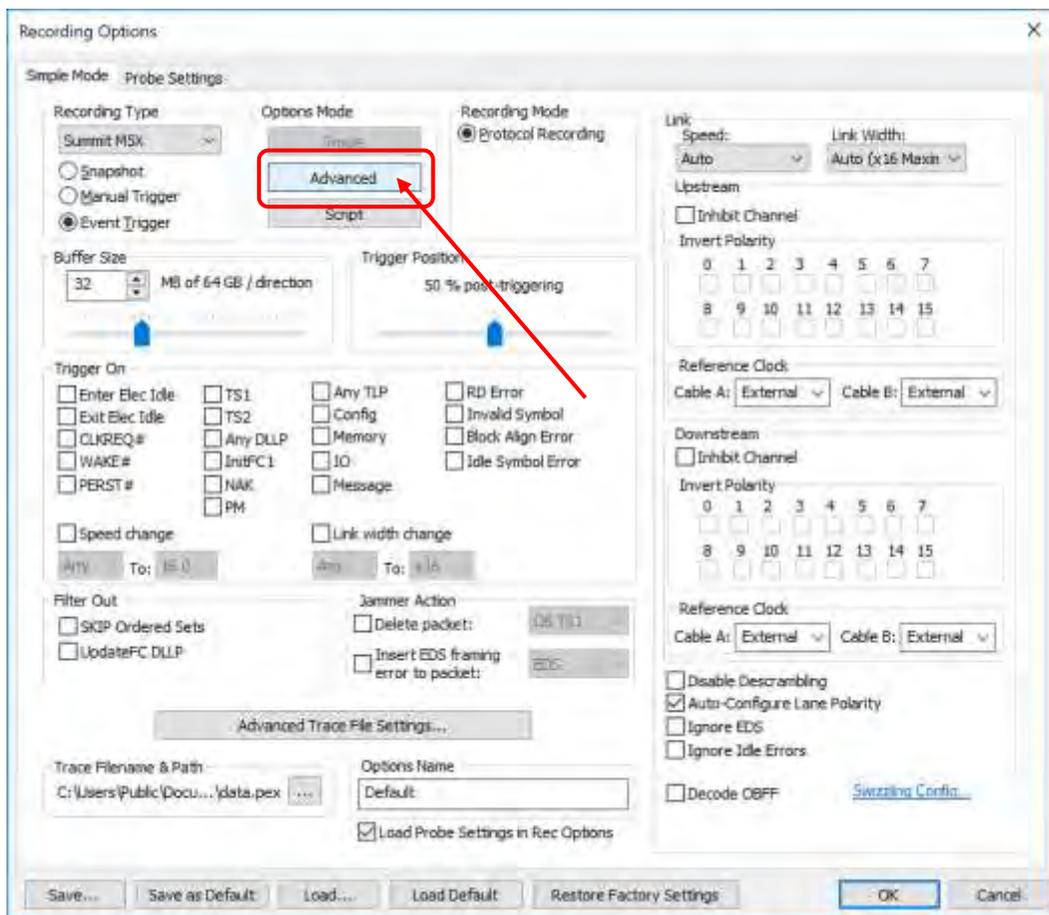


Selecting the “Setup” tab will pop up the following options dialog:

### 2.1.1 Recording Options

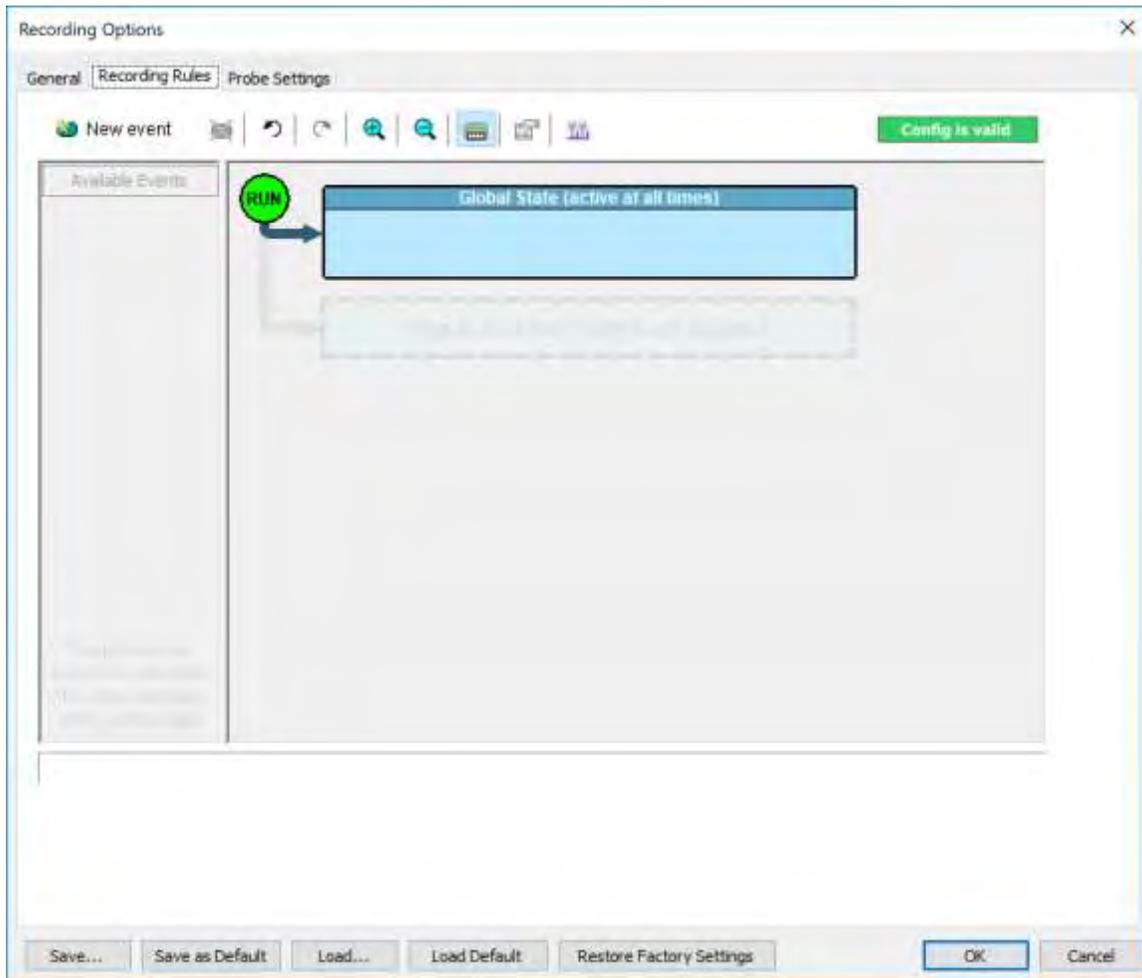


Select the Recording Options and a new dialog box will pop up:



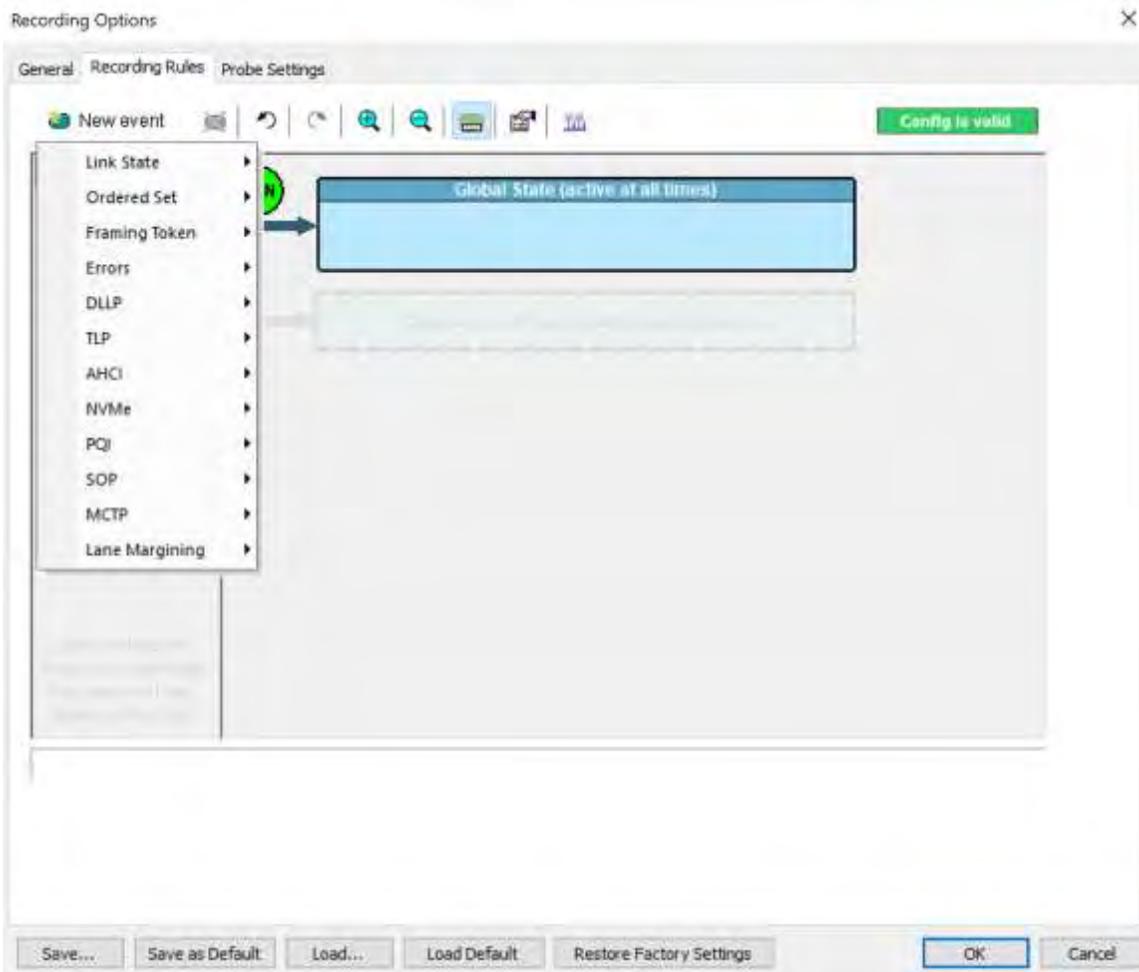
Select the Advanced Recording options and the Recording Rules tab will appear.

## 2.1.2 Recording Options: Recording Rules



If you select the New Event tab, you'll see all the options available to Trigger on.  
(see figure below)

## 2.1.3 Recording Rules: New Events



The PCIe Protocol Analysis software enables you to configure a very complex set of Recording Rules or events and Jammer Actions from the GUI and then convert them into a script.

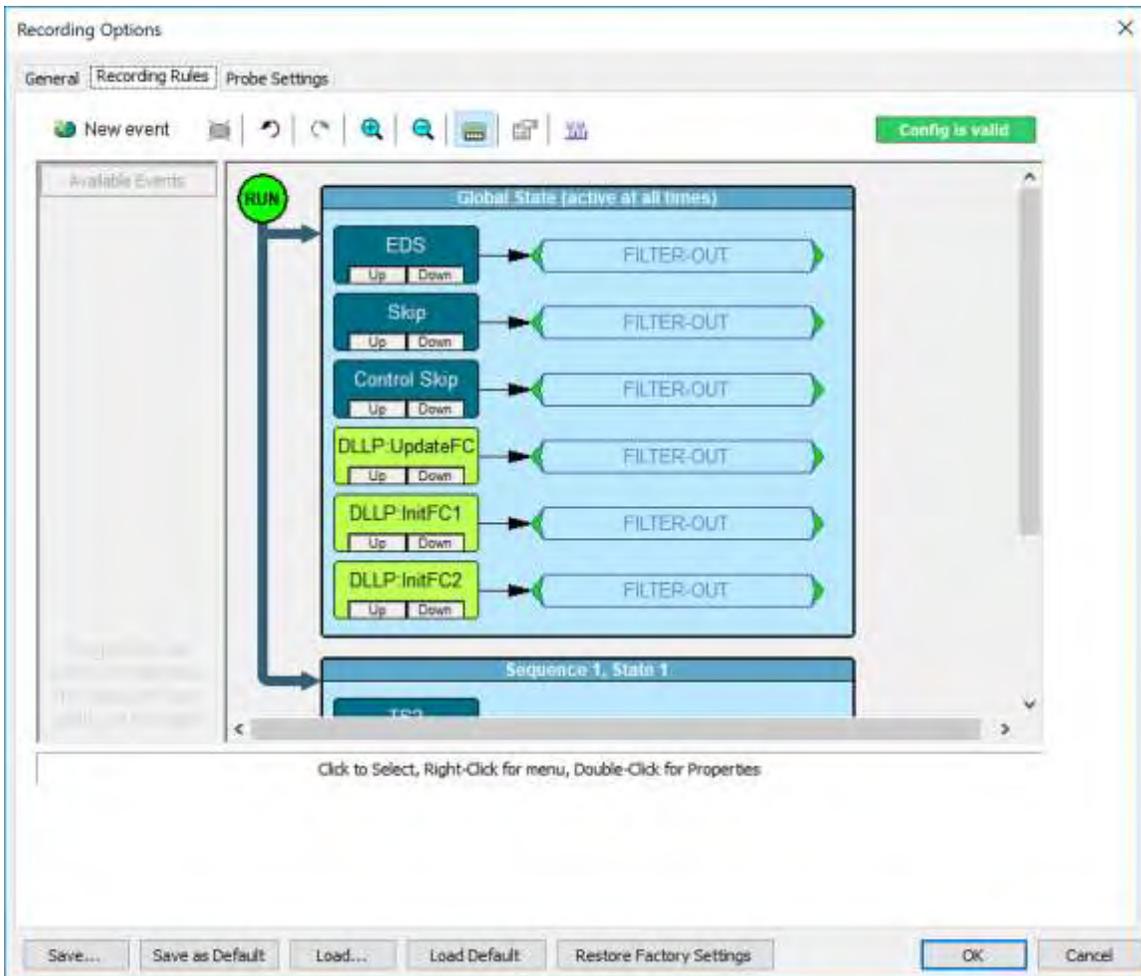
There are twelve (12) types of New events to choose from:

- Link State
- Ordered Set
- Framing Token
- Errors
- DLLP
- TLP
- AHCI
- NVMe
- PQI
- SOP
- MCTP
- Lane Margining

## 2.2 Converting Recording Rules and Jammer Actions from GUI to Script File

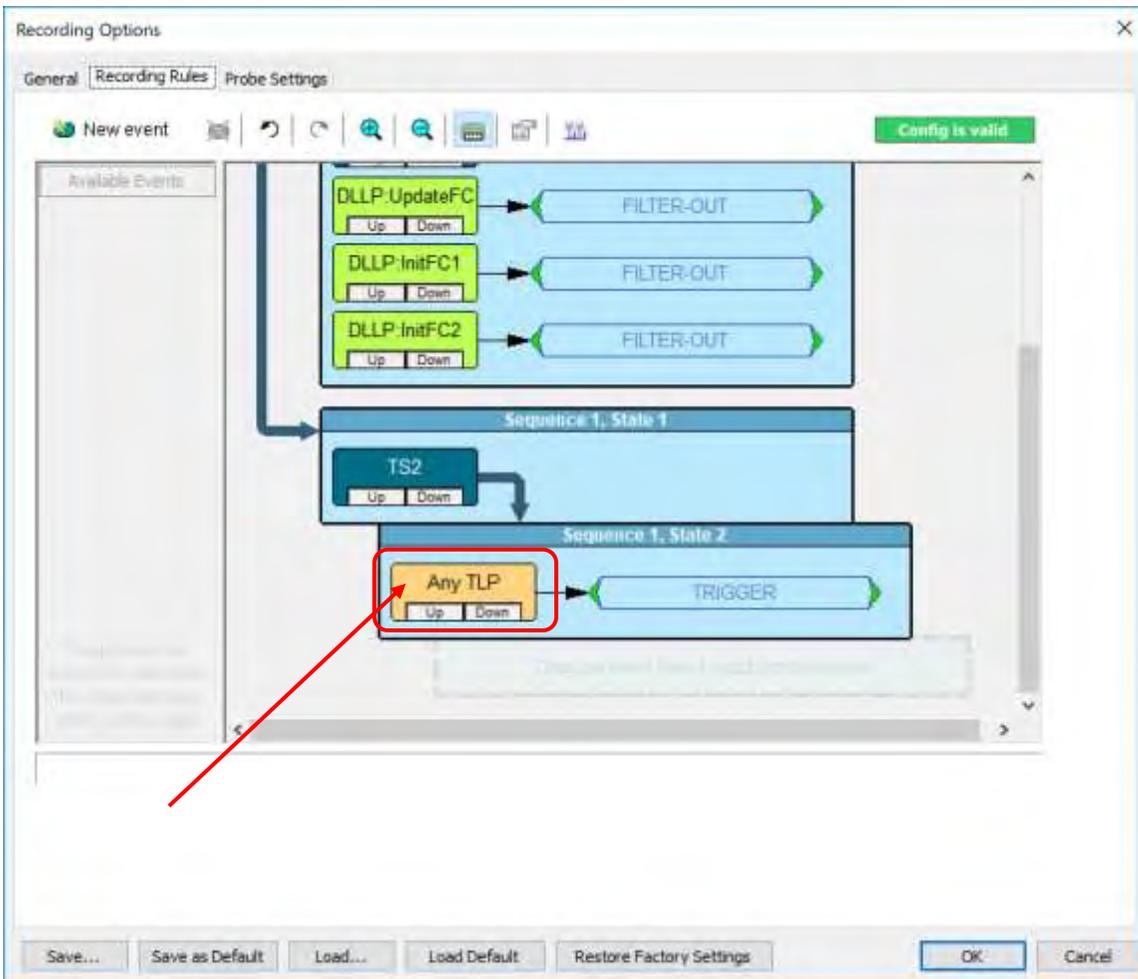
### 2.2.1 Complex Recording Rules (1)

We have defined a complex set of Recording Rules to which we can add Jammer Actions and then convert the GUI options into a script automatically.



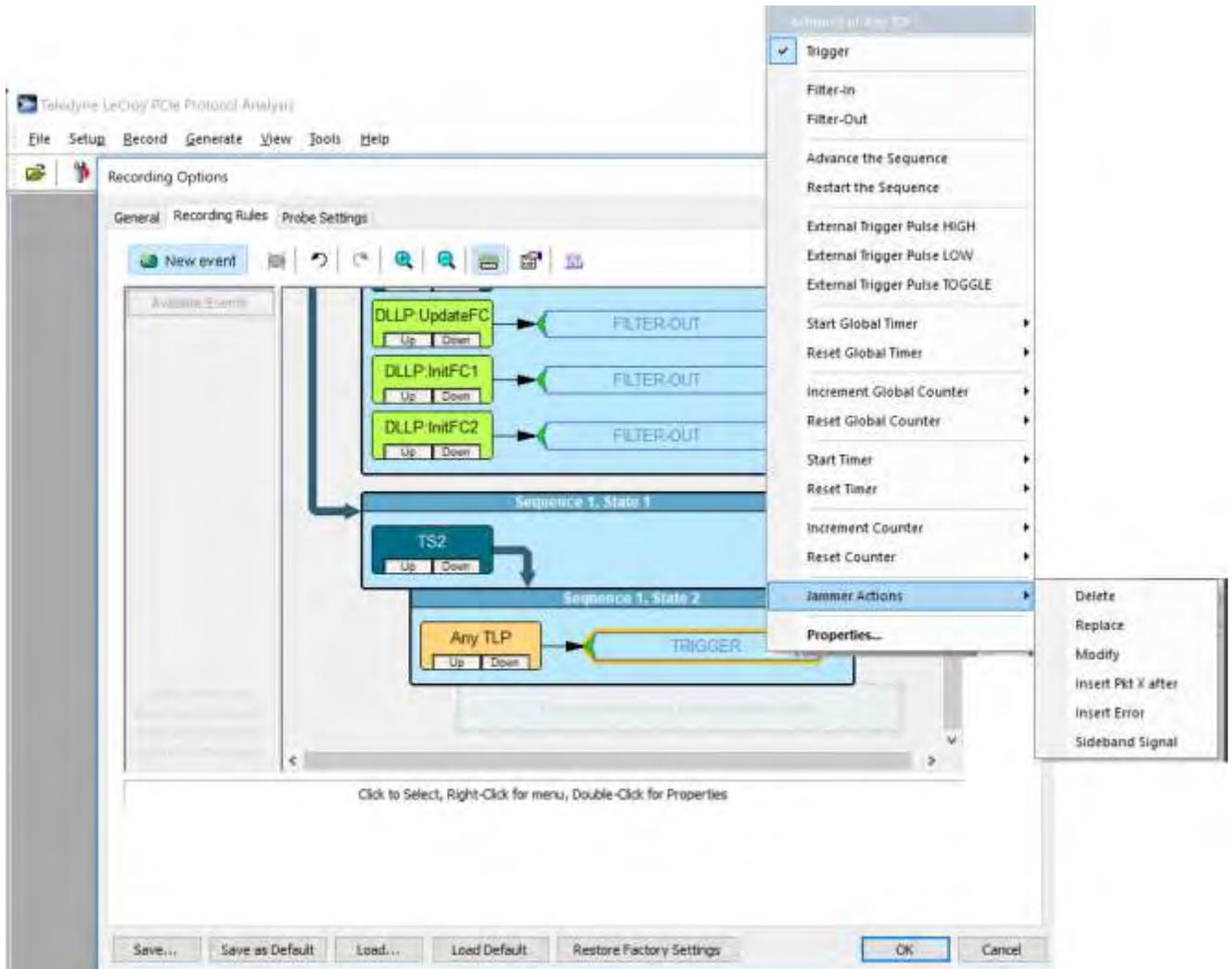
If you use the scroll bar on the right you'll get to the final events.

### 2.2.2 Complex Recording Rules (2)



If you right click on the Any TLP event another dialog will pop up which allows you to choose a Jamming Action as well as triggering the analyzer. See the figure below.

## 2.2.3 Jammer Options



Now you can add the following Jammer Actions to the Trigger:

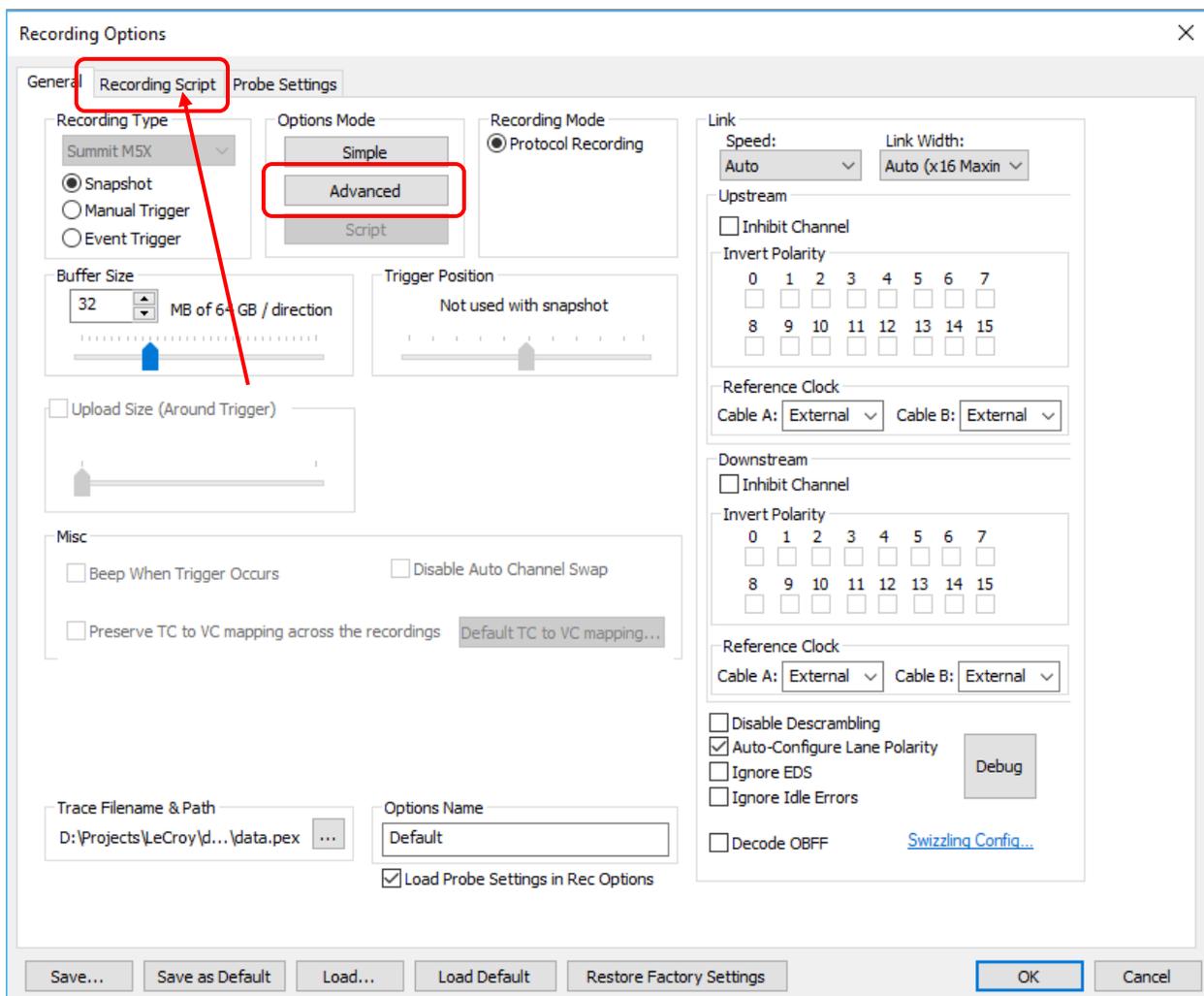
- Delete
- Replace
- Modify
- Insert Pkt X after
- Insert Error
- Sideband Signal

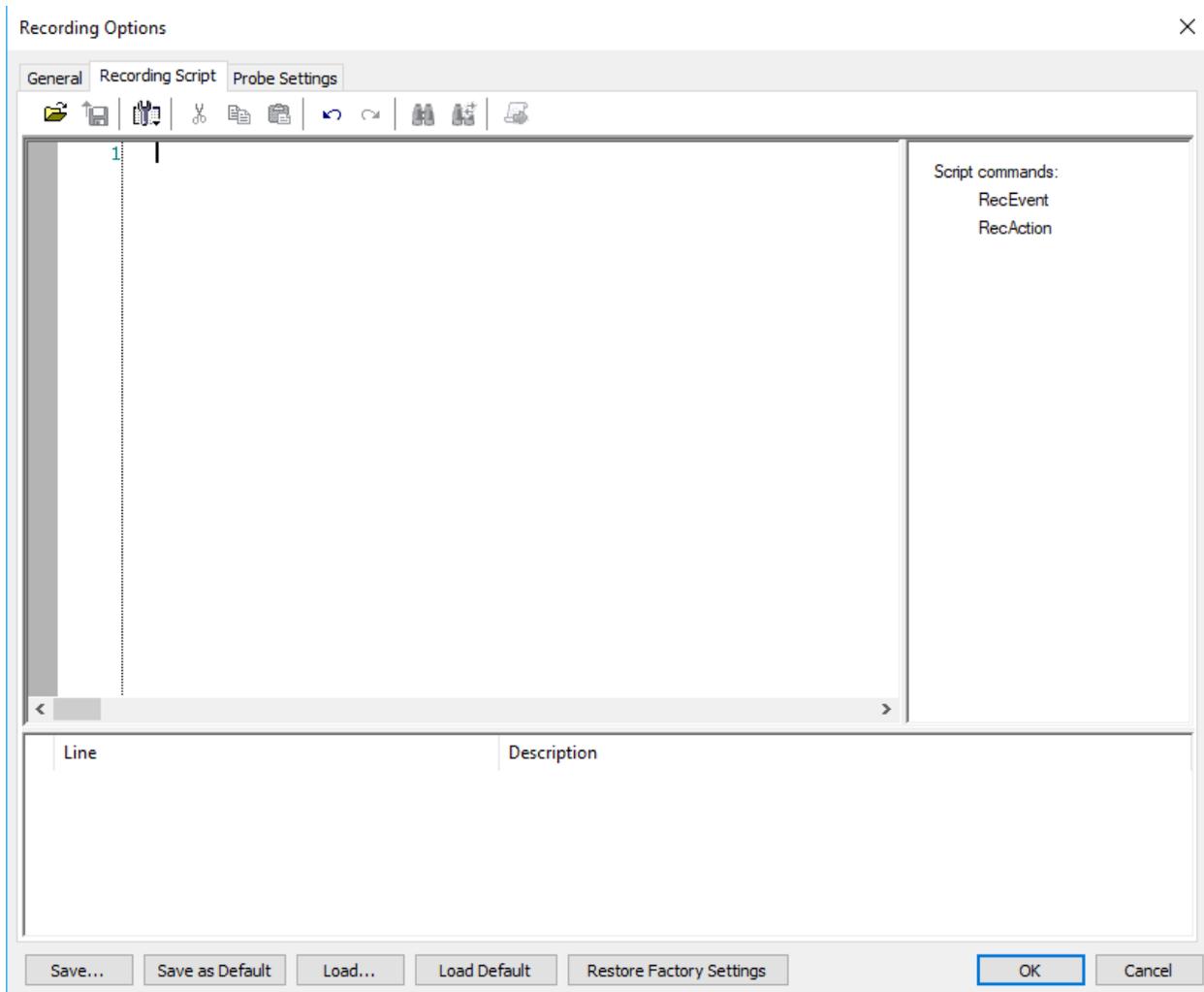
## 2.3 Script Mode

Script mode is currently available only for M5x devices. If an M5x device is not connected, then it is necessary to select “Summit M5x” device to enable script mode selection. Script mode can be enabled on the “General” tab in Recording Options dialog using the “Script” button. After switching to script mode, the “Recording Script” tab with controls to modify a script will be added.

If Recording Rules in Advanced or in Simple mode contain some events, these events will be automatically converted to a script when switching to script mode.

If Script Mode UI contains script text, then it will be converted to Recording Rules when switching from script mode to GUI mode.





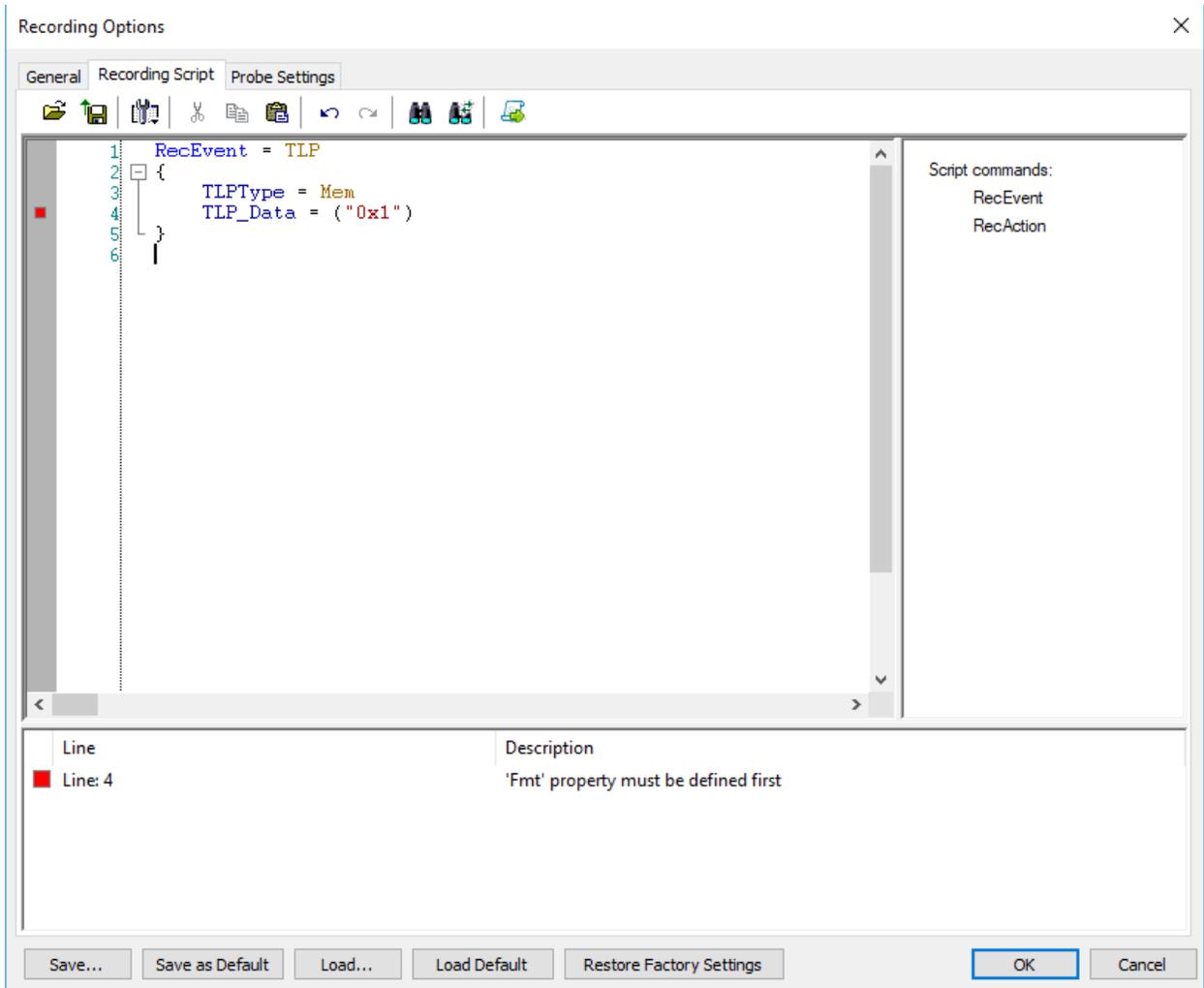
The Recording Script mode dialog contains: text area, panel with hints, error list and toolbar. Buttons in the toolbar include: Open/Save script, View Options of text area, Edit text (cut, copy, paste, undo, redo, find and replace) and Parse script.

Text for commands may be generated using menus "RecEvent" and "RecAction" on the right panel. They contain list of all supported events and actions. To generate text of commands it is necessary to click "RecEvent" or "RecAction" and select one of options from expanded context menu.

When the cursor (in the text area) is placed inside of braces of some command, then the hints panel will contain available properties for this command. See below.

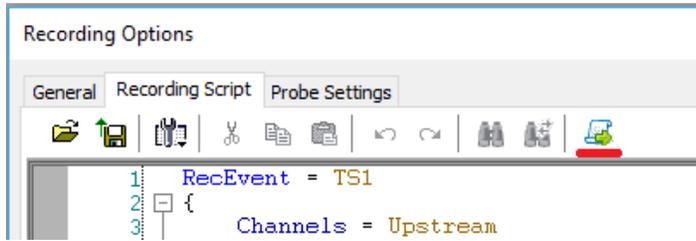
Parameter	Value
Channels	All
CurrentState	State_Global
Action	()
DLLPType	Update_FC
Credit_Type	P
VC_ID	P
Data_FC	NP
Data_Scale	CPL
Hdr_FC	"0x1"
Hdr_Scale	
DLLP_CRC	

The Errors list at the bottom will contain information (line number and description) about parse errors and warnings. It is possible to navigate to an error by double click on it.

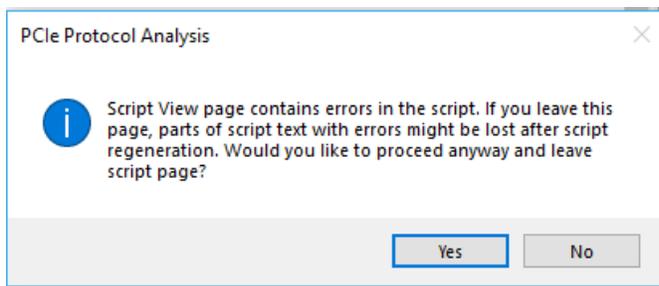


### 2.3.1 Script Parsing

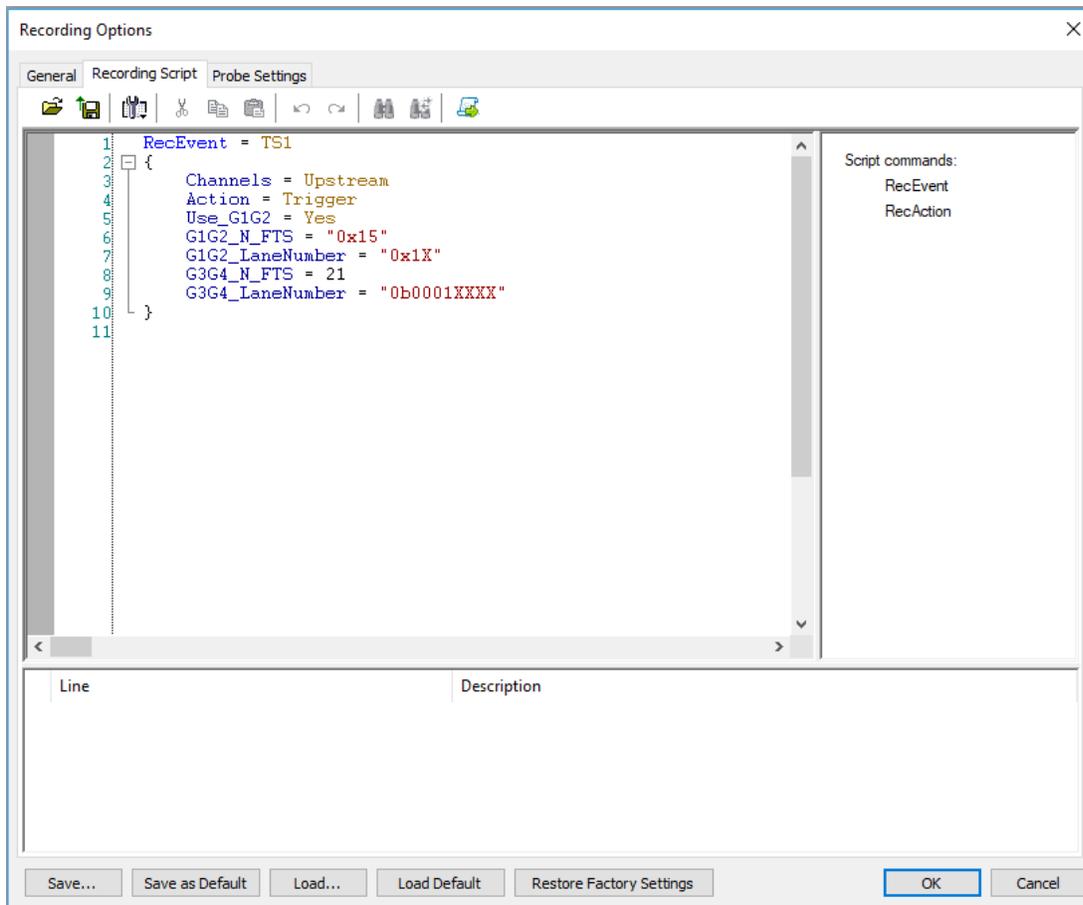
Script parsing can be performed by clicking “Parse script” button at the toolbar or by pressing Ctrl + S combination.



The Script will be parsed when leaving the script page or by closing the Recording Options dialog. If the script contains parse errors, then a corresponding warning message will be shown. Leaving a script page which contains errors may not create recording rules correctly. See warning message below.



## 2.4 Commands



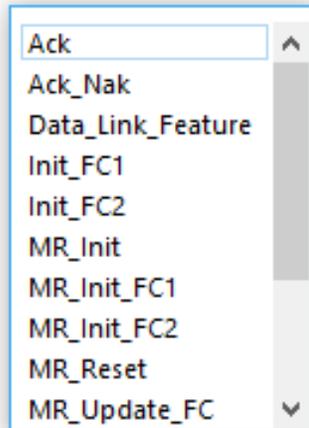
Script commands have properties, which are placed inside braces. Each property should be defined as a key-value pair in form “Key = Value”. Each command has its own set of available keys. There are different types of values: numeric, logical, mask/match and string. Logical values are represented as “Yes” or “No” keywords.

Mask/match values can be specified as decimal numbers or as strings with hexadecimal (“0x1”) or binary (“0b1001”) numbers. A mask can be applied using the “X” symbol (“0x111X” or “0b1001X111”). In binary notation “X” corresponds to 1 bit, in hexadecimal notation to 4 bits. If the value is specified as a string with hexadecimal numbers without mask, then the mask will not be applied to the whole field (for example, “0x1” equals “0x00000001” for 32 bit field).

Some properties have predefined constant values. Such values will be suggested when the “=” symbol is pressed. A list of such values can also be opened by pressing Ctrl + Alt + T or using context menu.

See example below.

```
RecEvent = DLLP
{
  DLLPType =
}
```



Some commands require certain properties to be defined. If such properties are not defined, then parse error will be added. If a command does not require a property and it is not defined, then a default value from UI will be set.

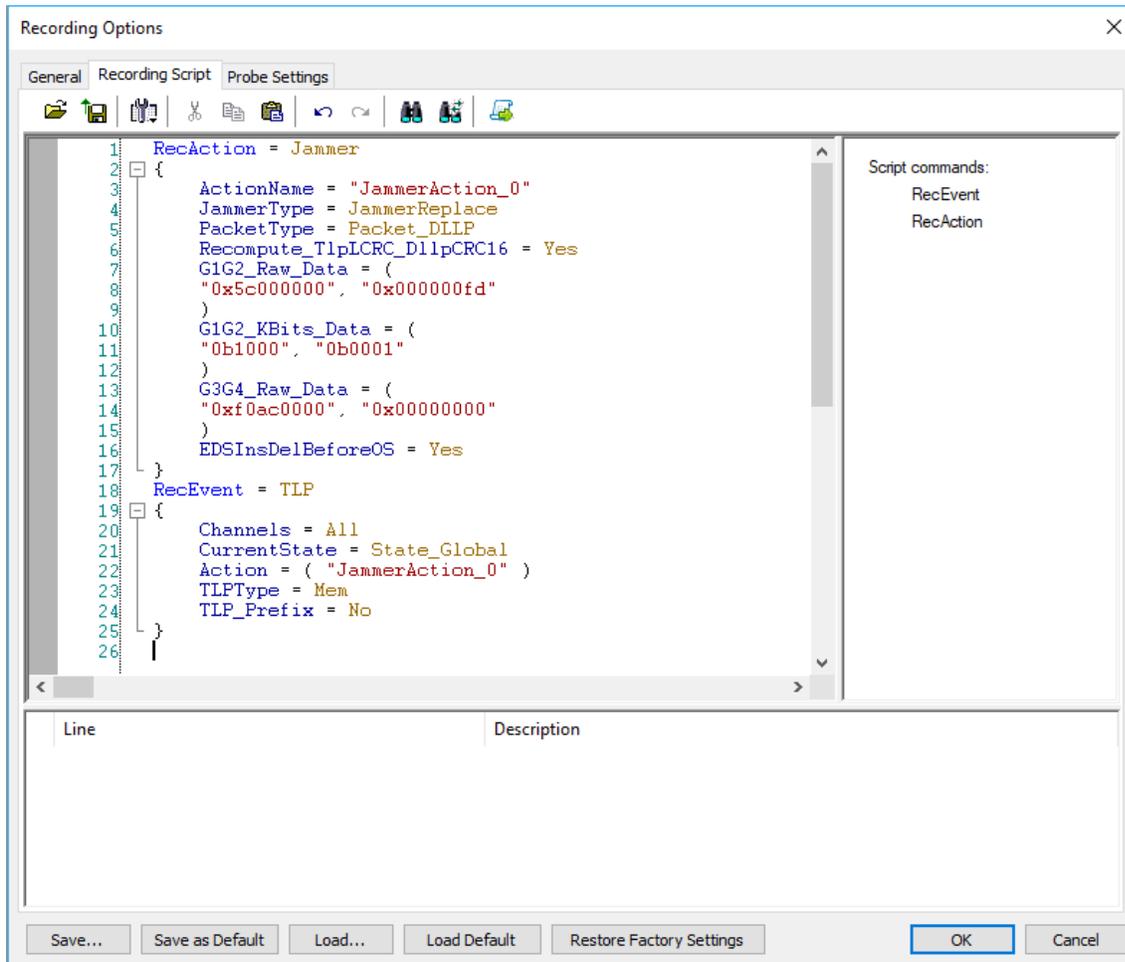
## 2.5 Events

RecEvent commands have several properties that are common almost for all events. Channels property defines event's channel(s): Upstream, Downstream, All or None. Current state property defines the state in which the event is located. It can be "State\_Global" or one of 16 local states (for instance, "State\_1"). Actions property defines the event's actions, which is used for triggering, sequencing, filtering etc. Action property is also used to connect events with jammer actions.

Timer and counter events have additional restrictions. They are divided into two groups: global and local. Global timers and counters can be created only in global state. Local timers and counters must be defined in the state property and cannot be created in global state.

A list of available properties may depend on the values of other properties. For example, TLP event's properties depend on TLP type.

## 2.6 Jammer Actions



Jammer Action is a type of script RecAction command. The Jammer Action command must be connected with an event or it will not be created. Each jammer action must have a unique name and its type must be defined. Jammer Action can be defined only before event in which it is used. To connect an event with a jammer action it is necessary to add a string with the jammer action name to the list of the event's actions.

If the Jammer Action modifies the raw data property and is not defined, then raw data will be the same as event's layout.

If raw data is not defined for Replace or Insert Pkt XAfter jammer actions, then raw data will be set to the default value for the Jammer Action's event type.

## 3 Scripting Jammer Functionality

There are seven (7) types of New events (that are supported) to choose from:

- Link State
- Ordered Set
- Framing Token
- Errors
- DLLP
- TLP
- NVMe

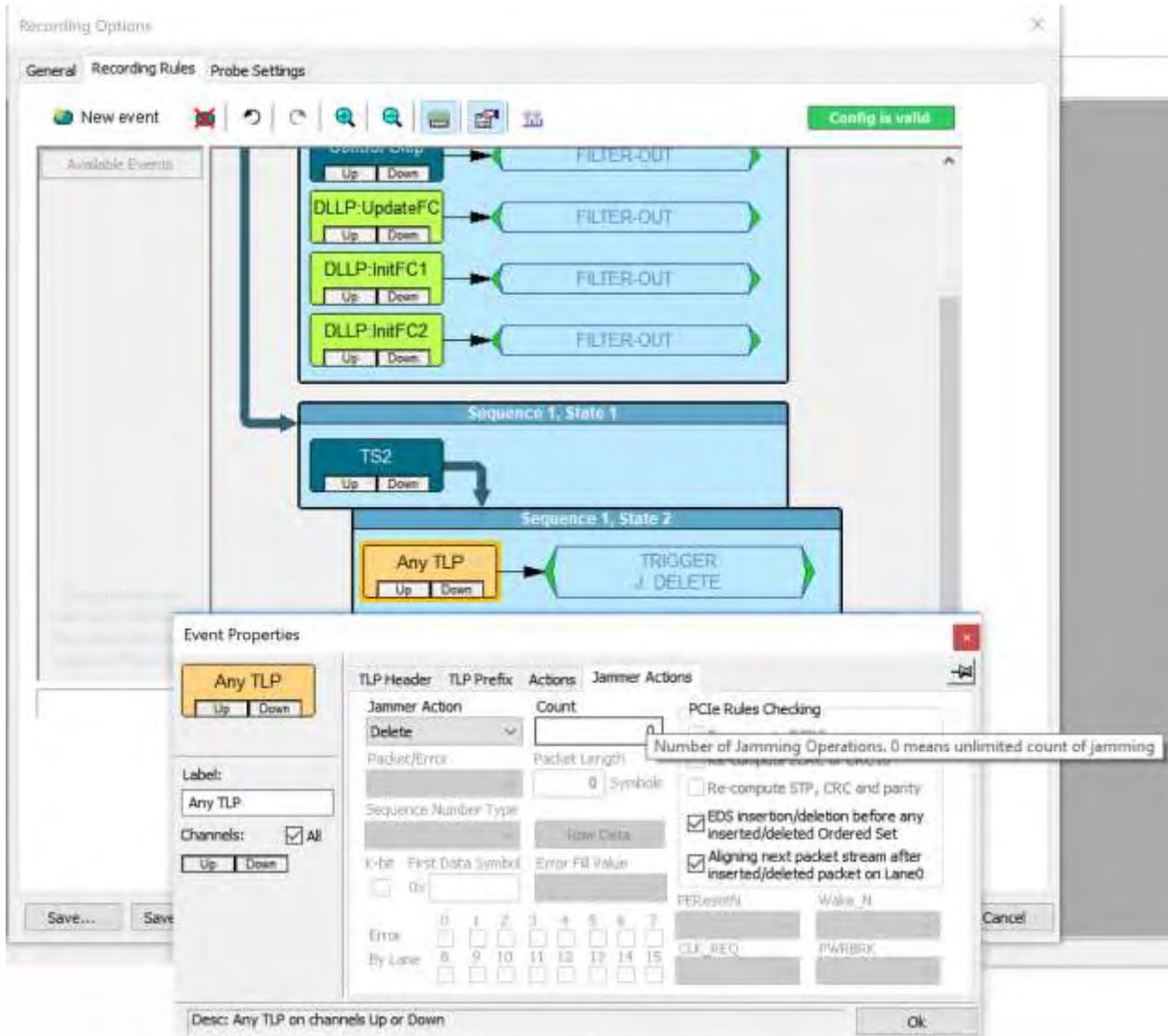
There are six types of Jammer Actions available which can be associated with any of the New event types list above:

- Delete
- Replace
- Modify
- Insert Pkt Xafter
- Insert Error
- Sideband Signal

It is allowed to create any of six jammer actions and assign it to any event in script text. But if jammer action is not applicable to event (after it is assigned to event) – there will be error after script saving/compilation indicating that such jammer action cannot be assigned to event.

### 3.1 Example: Select Jammer Action: Delete

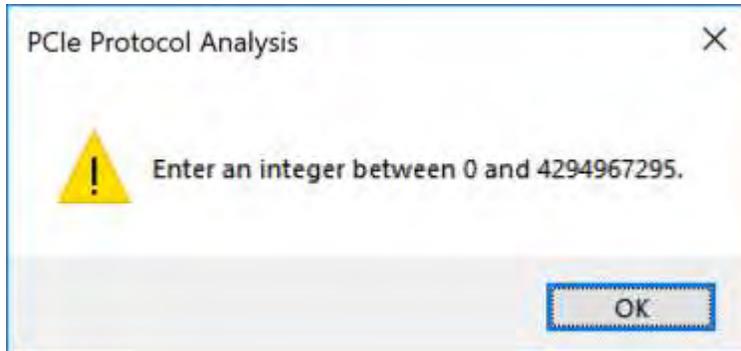
If you select Delete as the Jammer Action the following dialog will pop up showing you the Jammer Action options.



**Note:** If you set the Count = 0 the Jammer action will be repeated an unlimited number of times. This will effectively stop the script from proceeding any further.

For the Delete Action there are only three extra options:

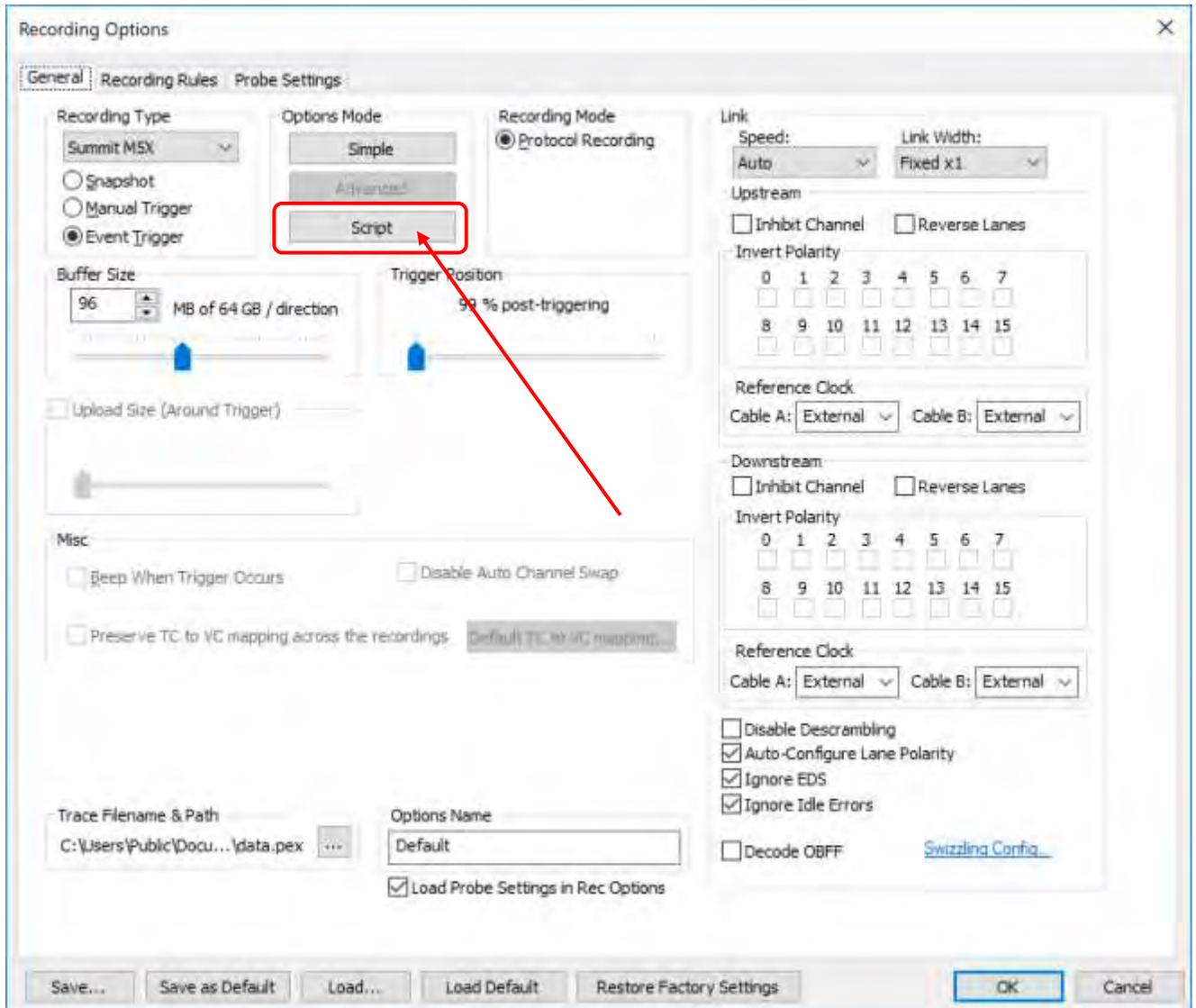
- Count: Number of Jamming Operations -> 0 means unlimited
- If you enter too large a number the following warning message will pop up



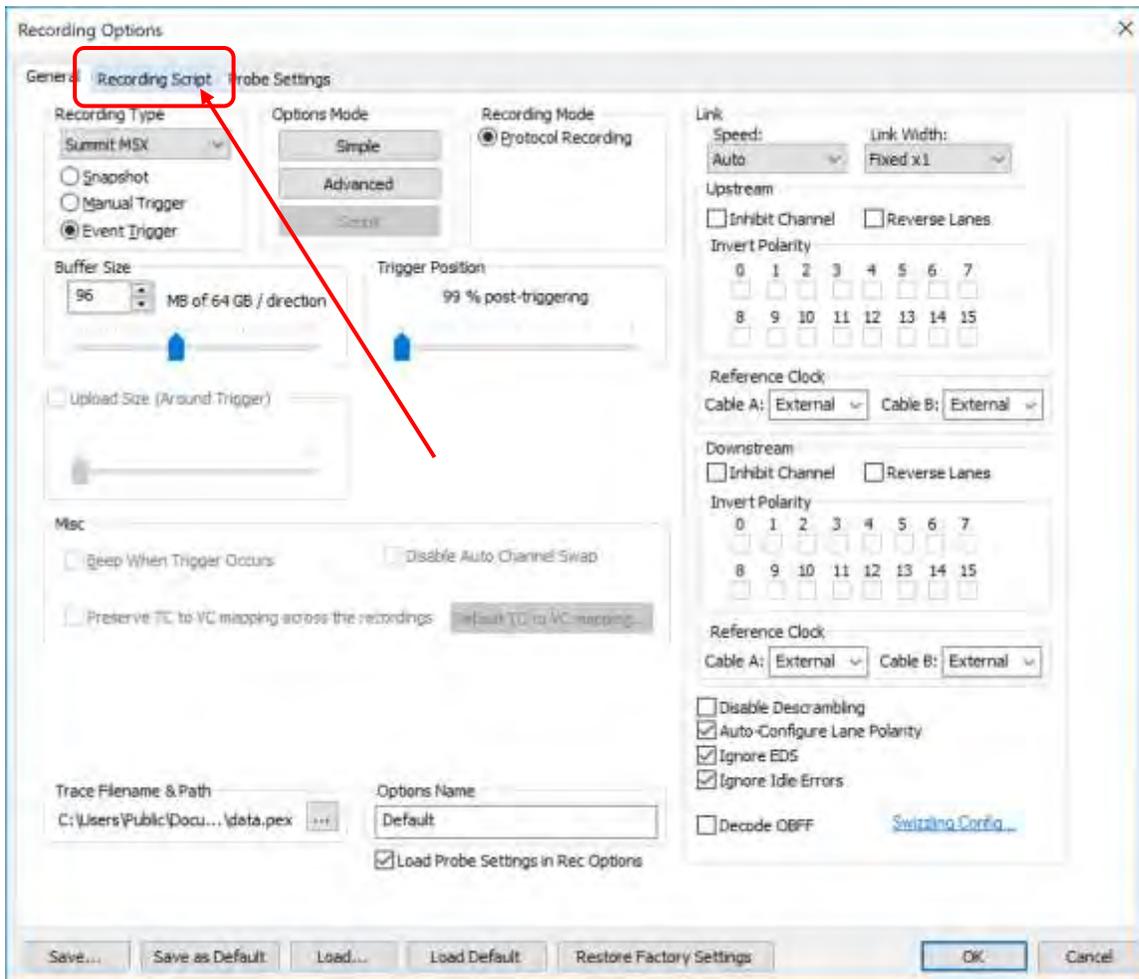
- For this example enter 10
- EDS insertion/deletion before any inserted/deleted Ordered Set
- Aligning next packet stream after inserted/deleted packet on Lane0

Click on OK to add the options to the Jammer Action.

If you select the General Tab (in Recording Options) the following dialog will appear.

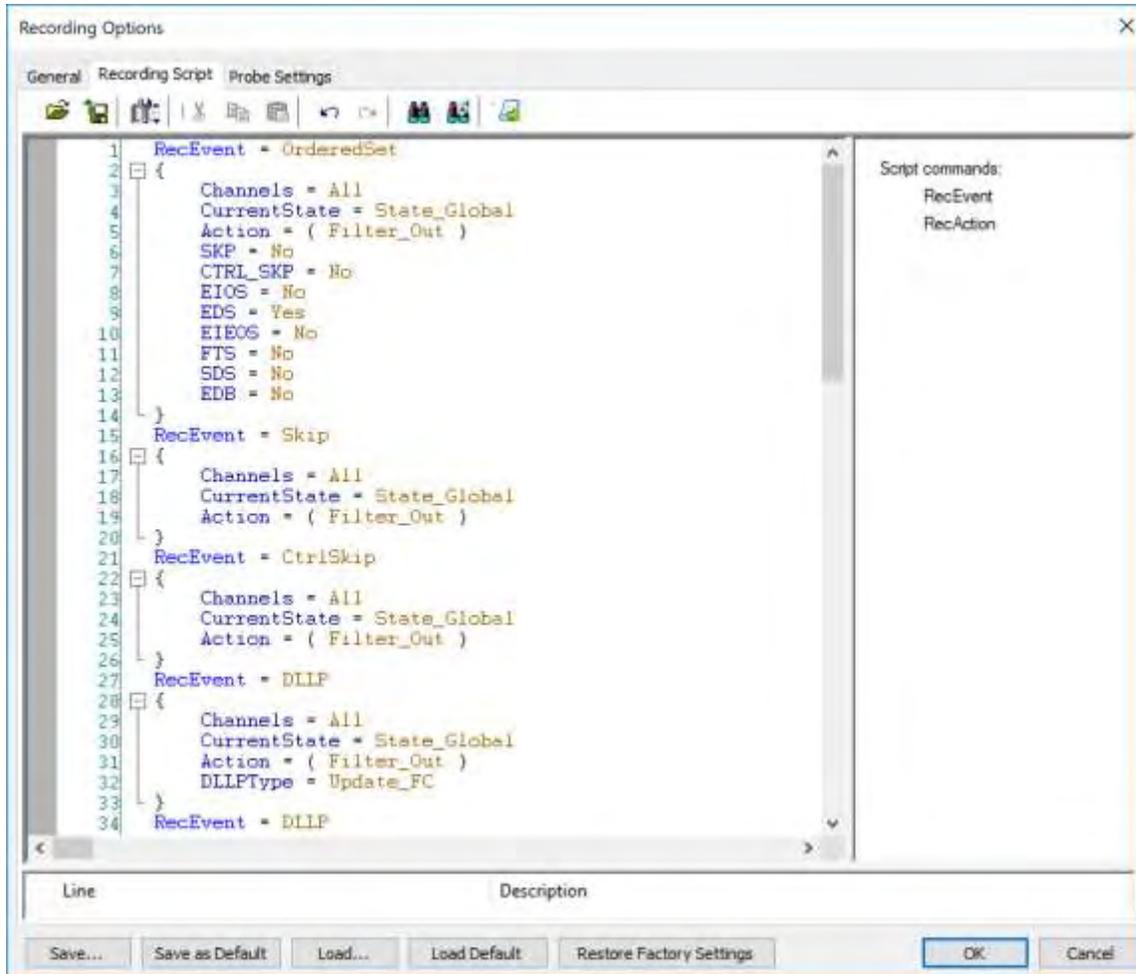


If you select the Script Option under Options Mode the Recording Script Tab will appear in the Recording Options -> General dialog (see below).

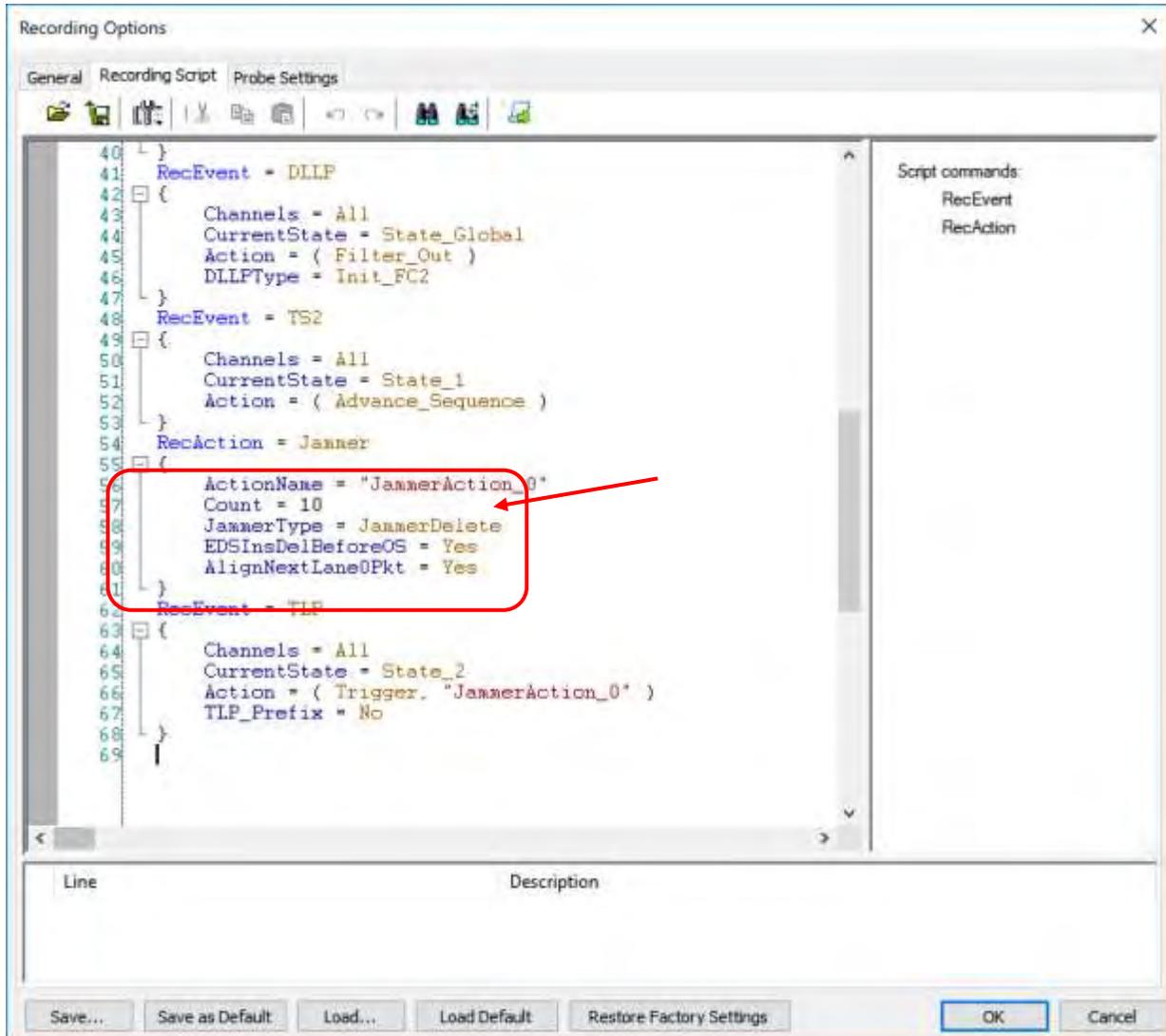


If you select the Recording Script tab the script to generate the complex Recording Options and Jammer Action will appear. See below.

## 3.2 Recording Options and Jammer Action Script

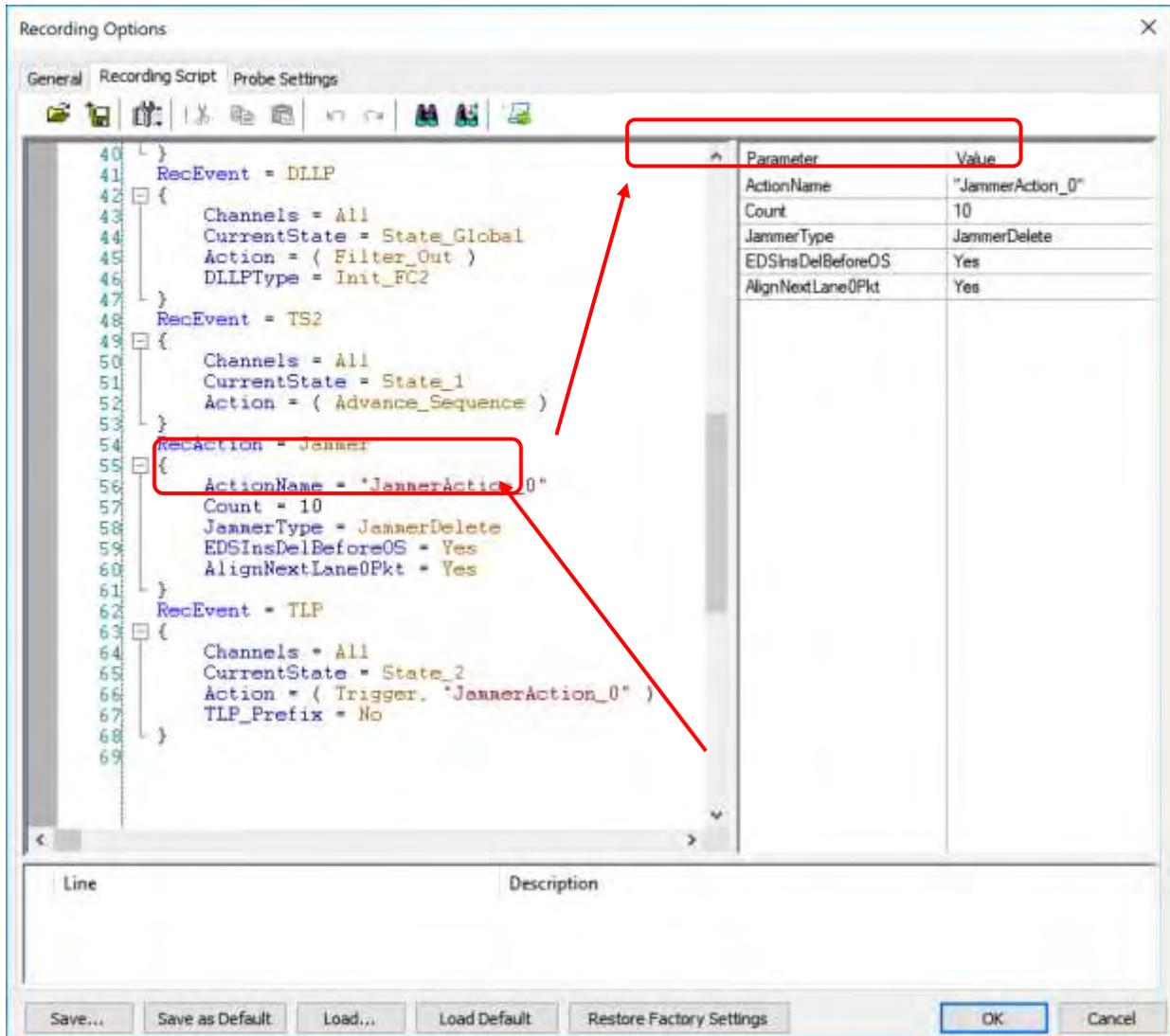


If you scroll down you'll see the rest of the script.

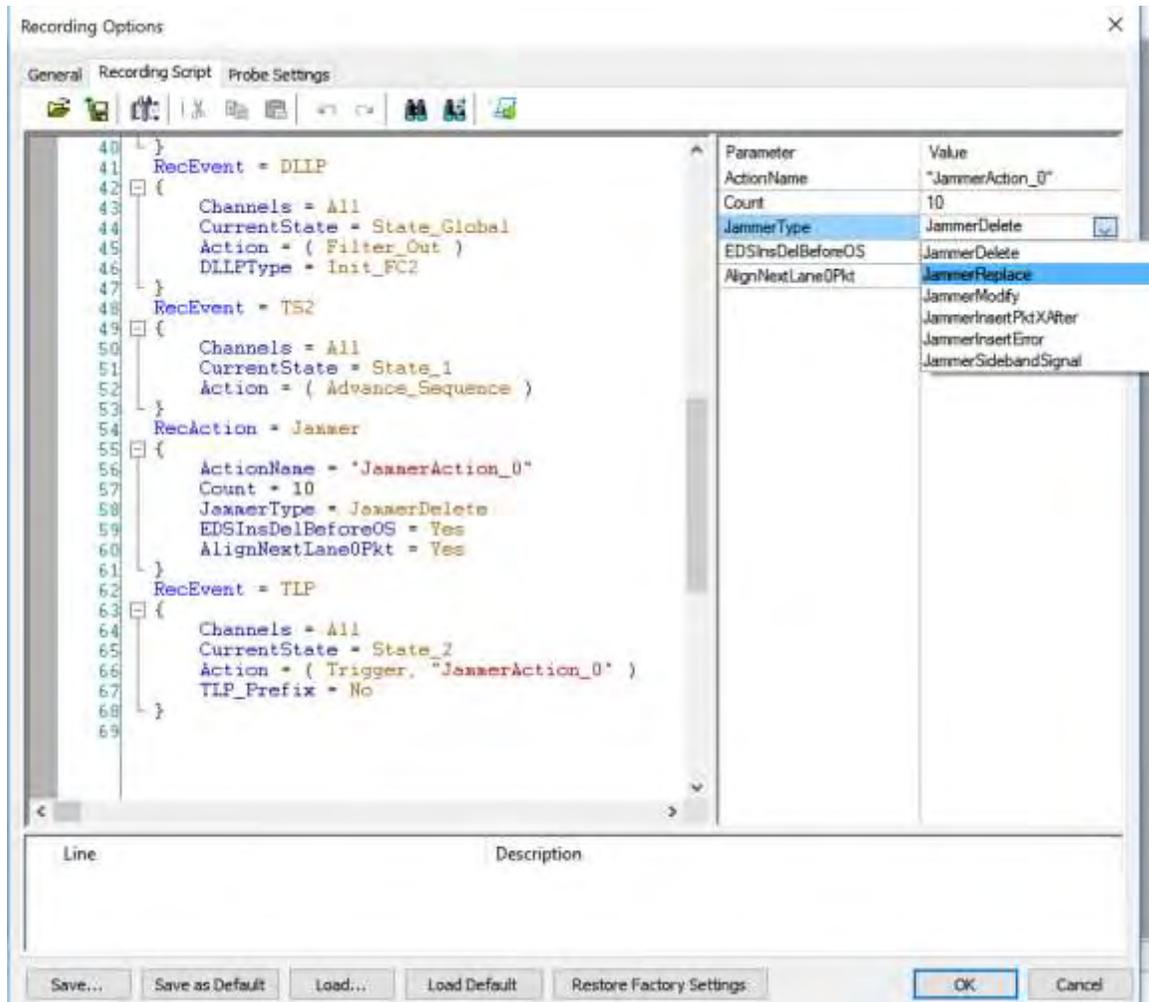


If you select the ActionName = "JammerAction\_0" the details of the Jammer Action will appear on the right side of the dialog. See below.

### 3.3 Updating Jammer Action Script

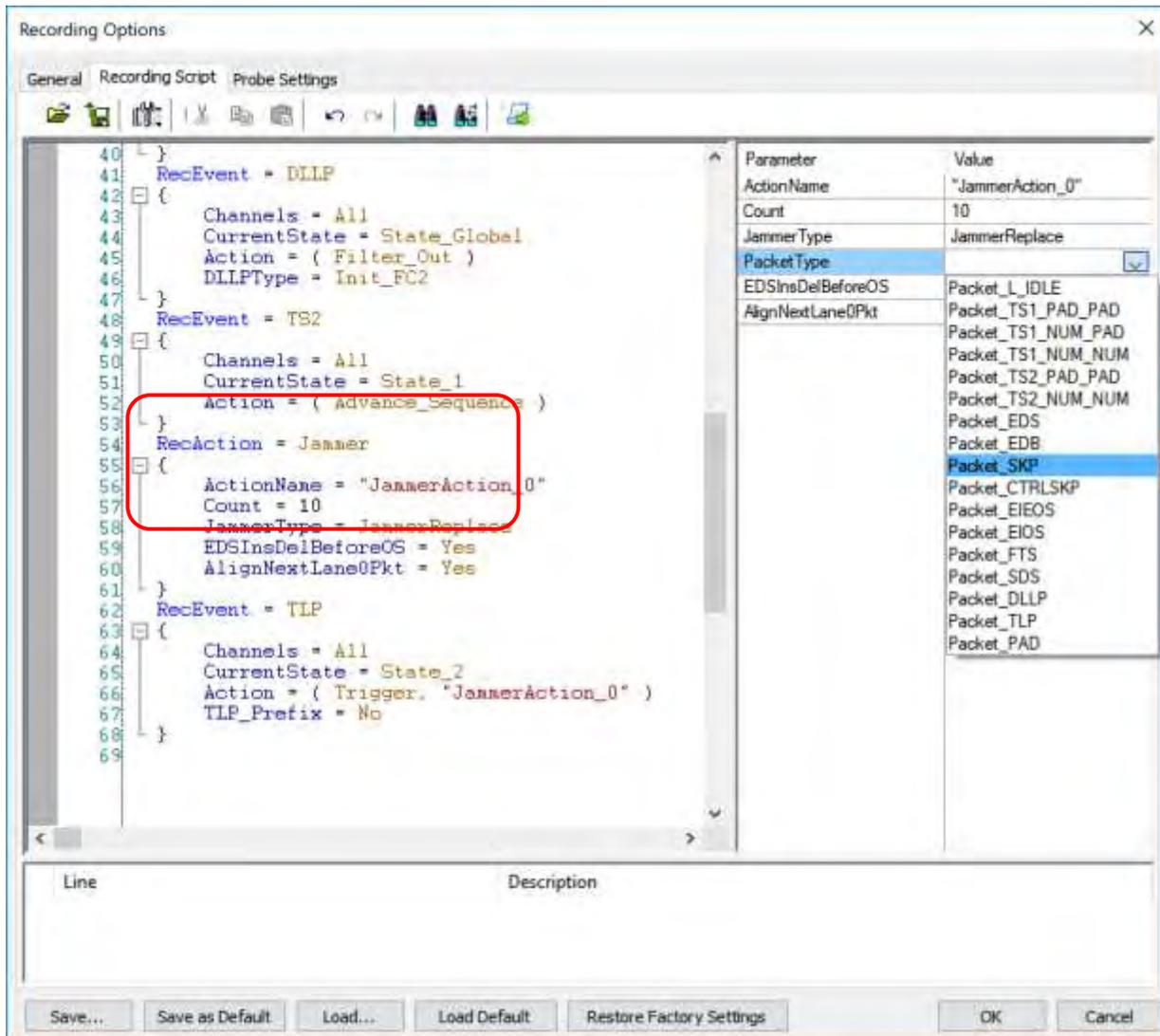


You can modify the script by selecting a Parameter Value as shown below.

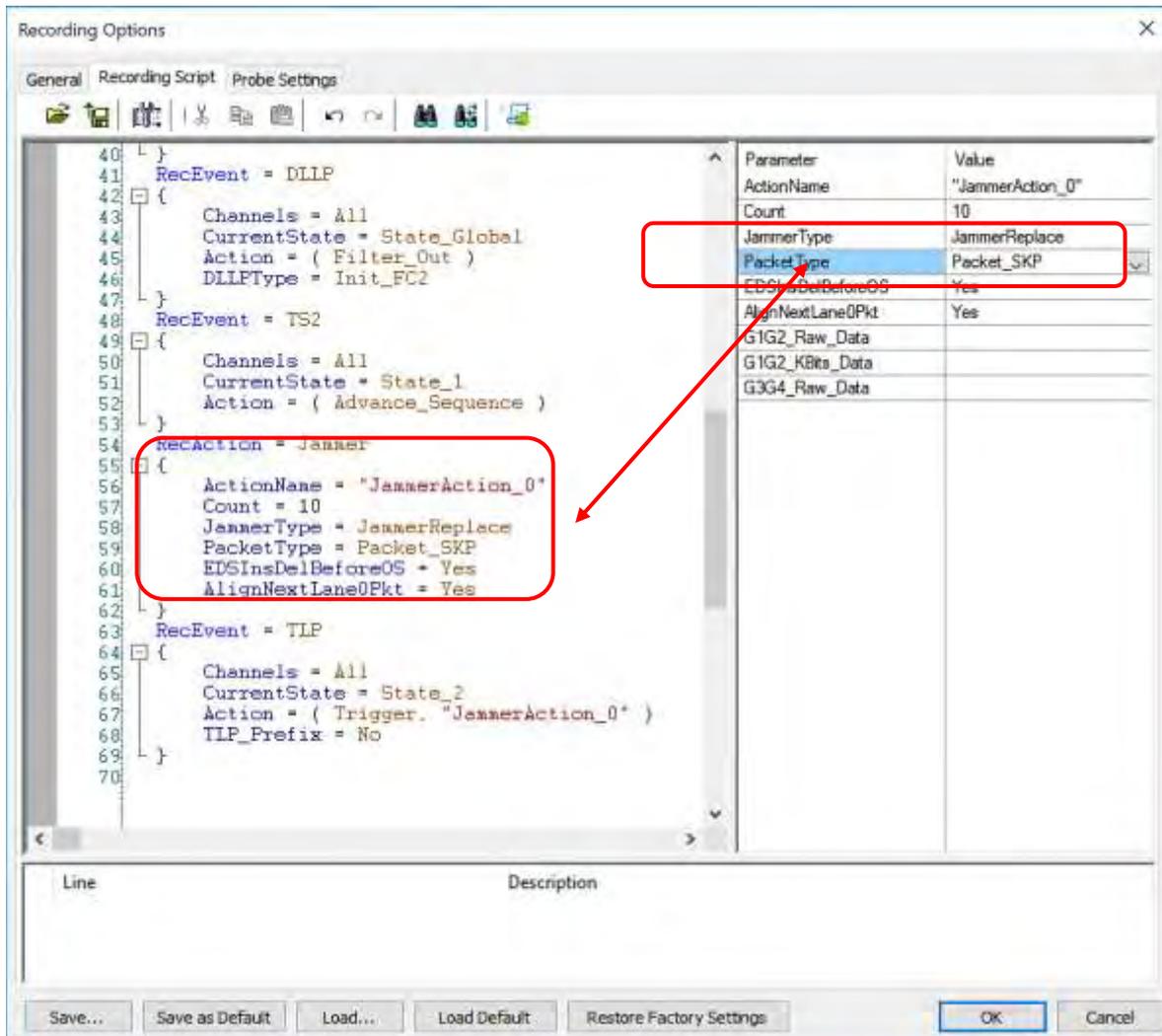


If you want to change the Jammer Action, just select the down arrow next to Jammer Delete and the other types of Jammer Actions will be available.

Select JammerReplace and the options for that Jammer Action will appear. See below.

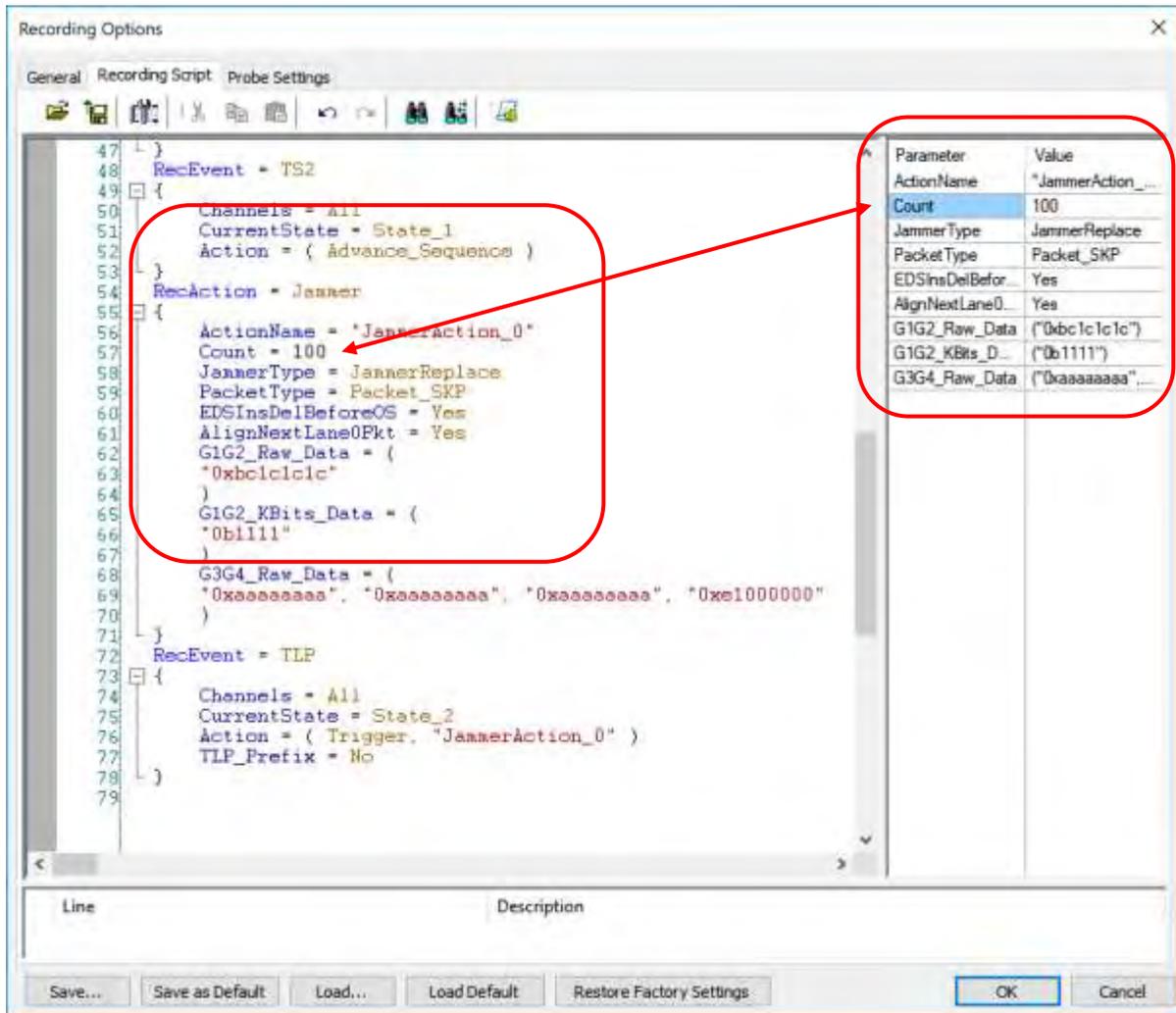


After Packet\_SKP has been selected the script will be updated. See below.



You can also modify the Count from 10 to 100.

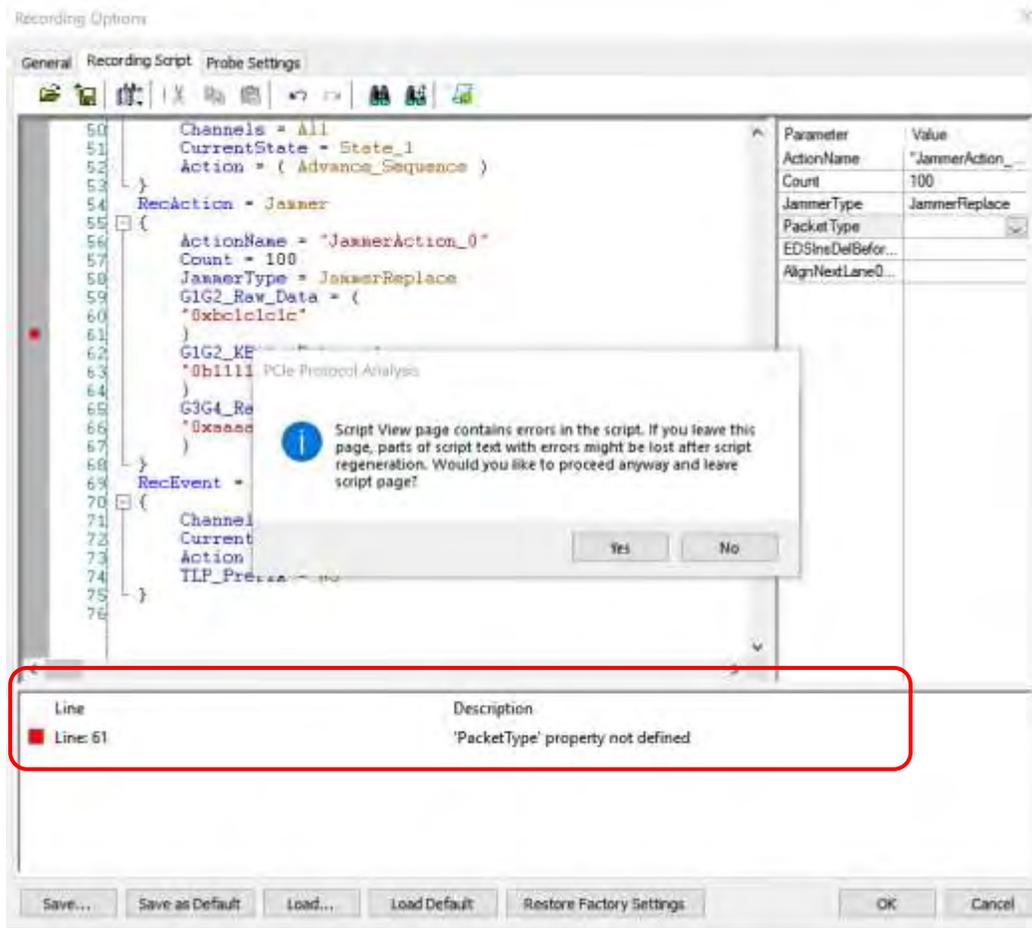
Simply enter 100 into the Value block and move your cursor into the Count tab and click on the left mouse button. These actions will update the Count in the Script to 100. See below.



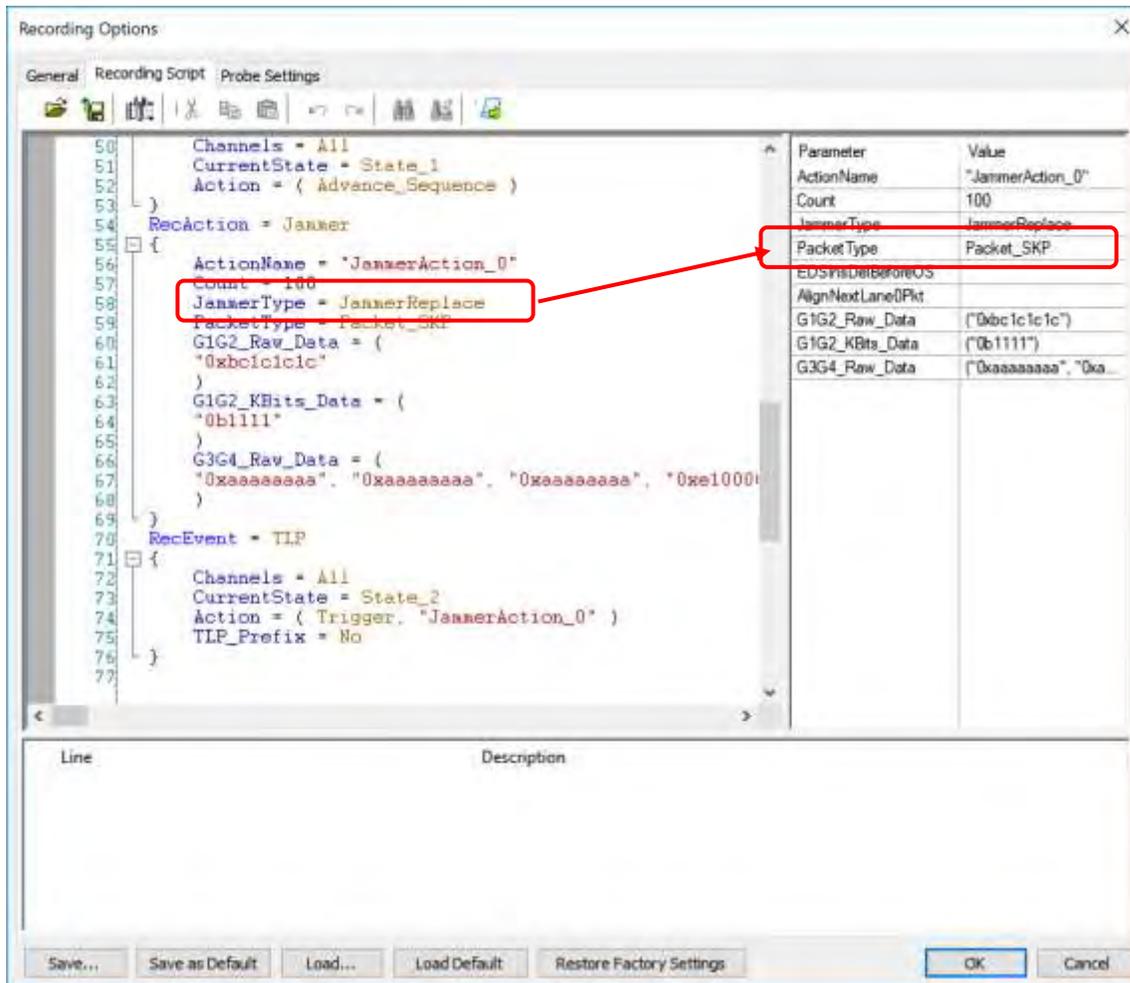
Now that you've modified the script you can select the General tab and return to the GUI format. See below.

## 3.4 Warning Message if Mistake in Jammer Script

If you make a mistake in the script, you'll see the following warning message and how to correct your mistake.

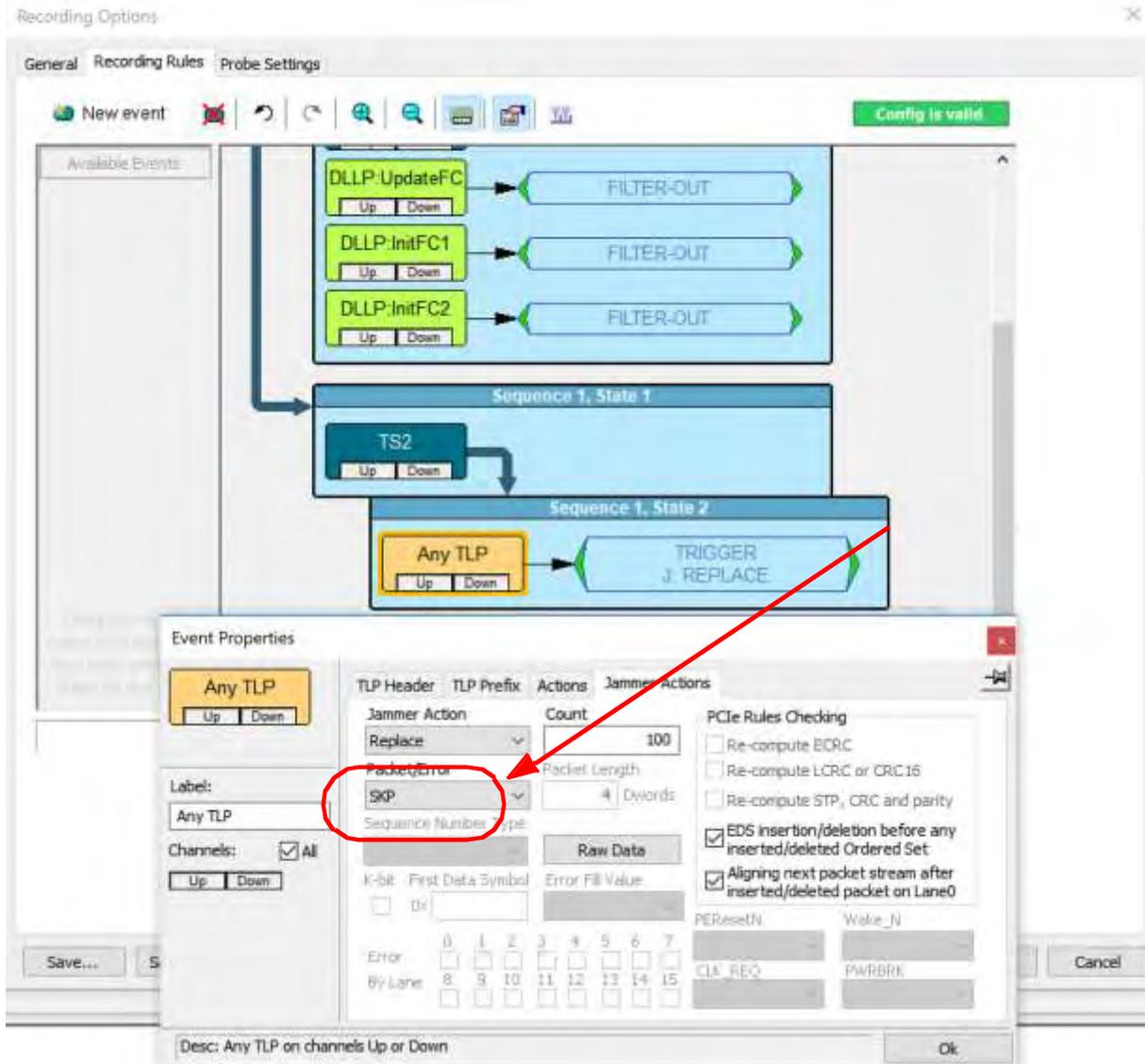


On Line 61 the Packet Type must be defined. Select No and correct the error. Once you do that and click "OK" you can return to the Recording Options -> Recording Script and see that your script has been modified and is correct. See below.



Now you can select the General tab and go back to the GUI. You'll see the GUI has been updated to the changes you made in the script. See below.

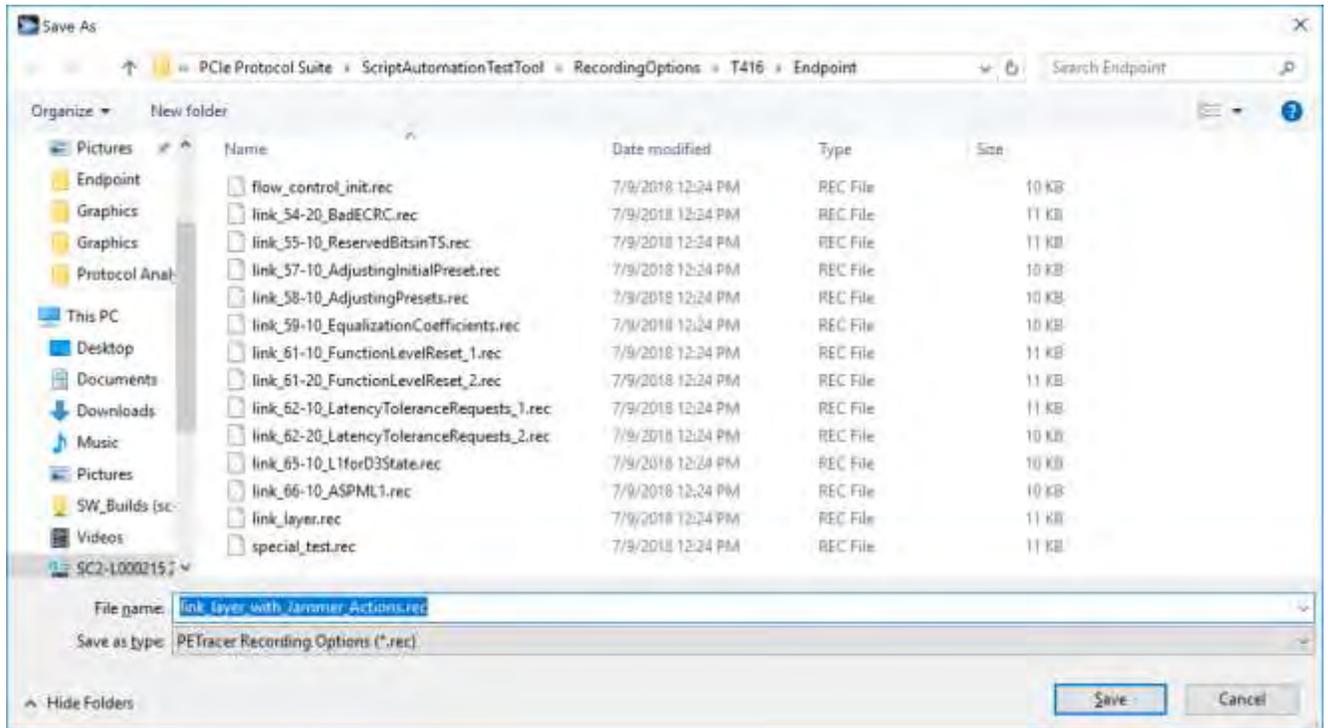
### 3.5 Updated GUI After Scripting Mistake Corrected



You can see that the Jammer Actions have been changed with a Packet of Type SKP.

## 3.6 Saving a Jammer Actions Script File

Another option you have is to Save the script file you've created with the Jammer Actions. See below.

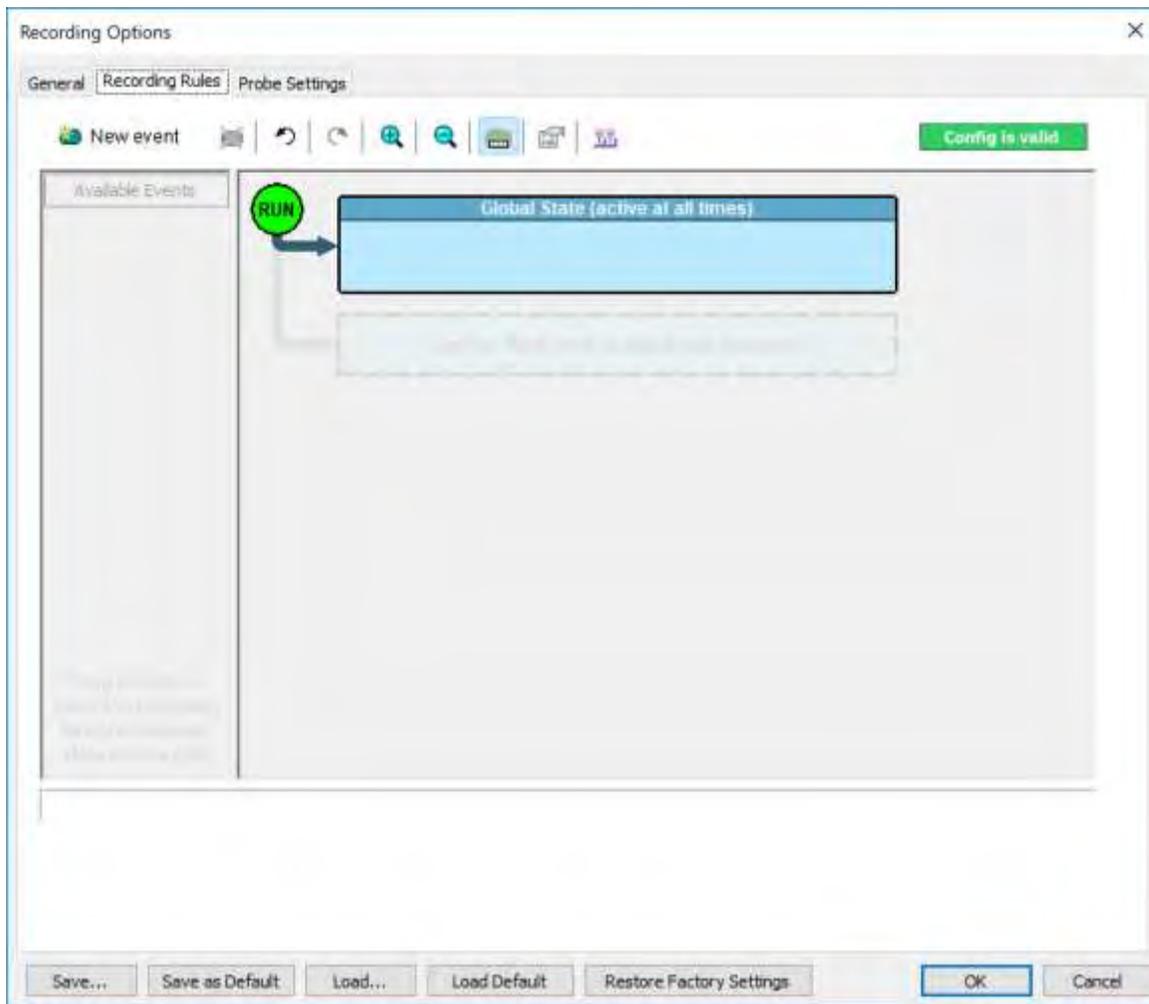


Click on "Save" and the new Jammer Actions script will be saved, to be used in the future.

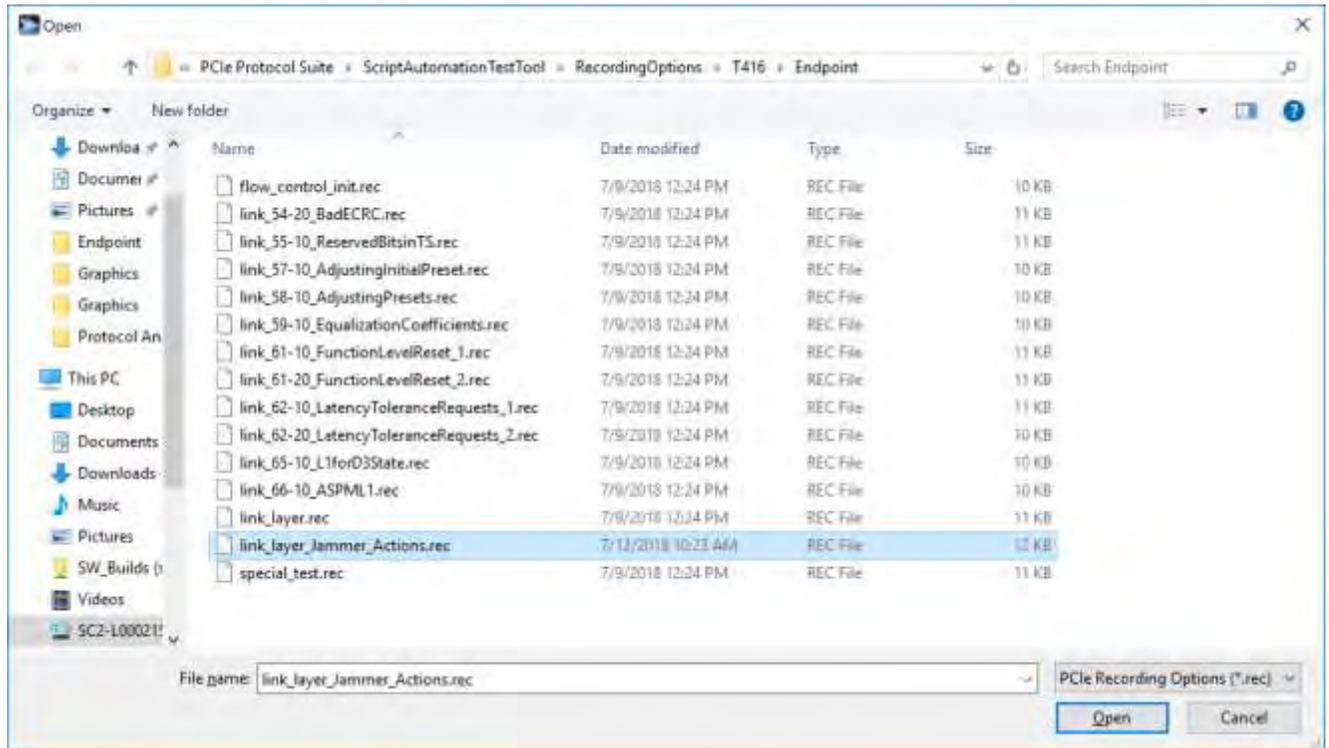
## 3.7 Loading a Jammer Script File

To load the Jammer Actions script file go to Setup -> Recording Options. Choose the M5x and select Recording Rules.

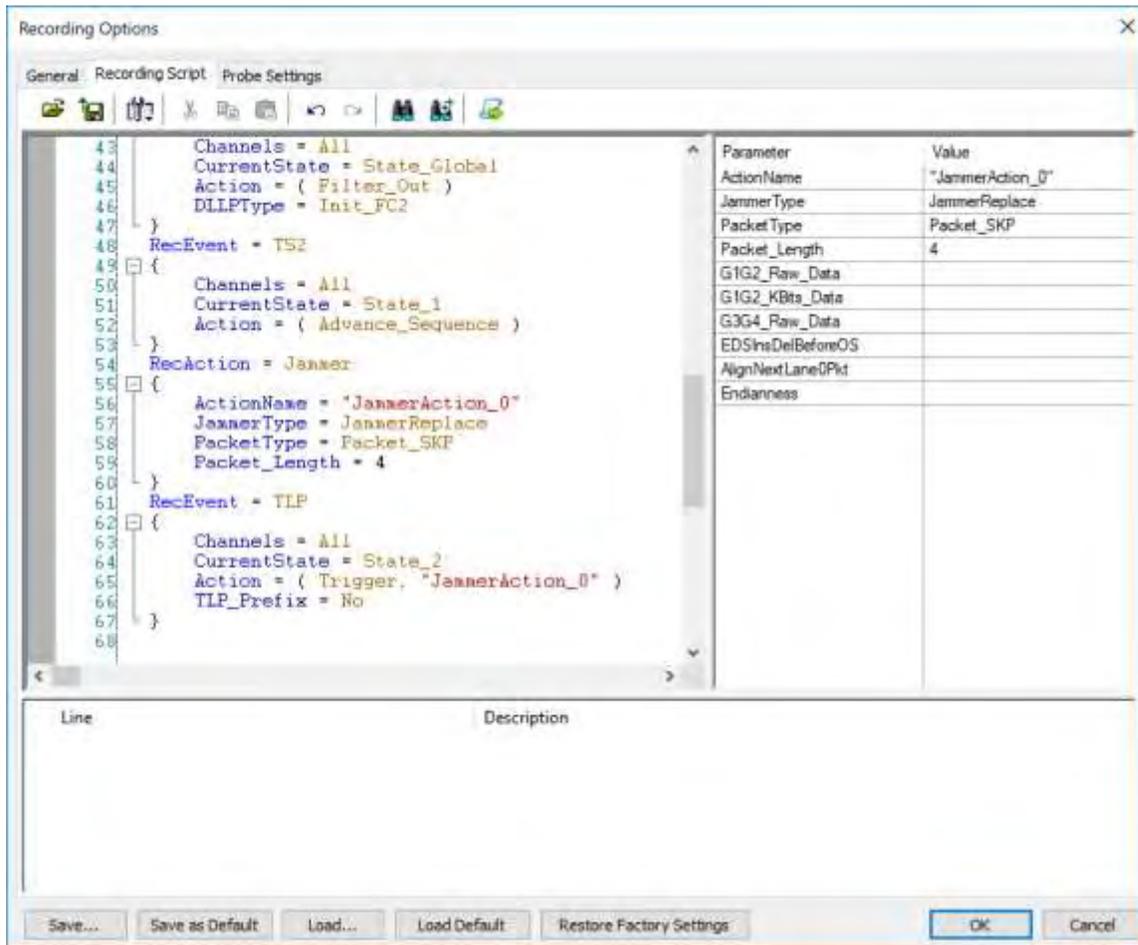
Then select "Load" (see below)



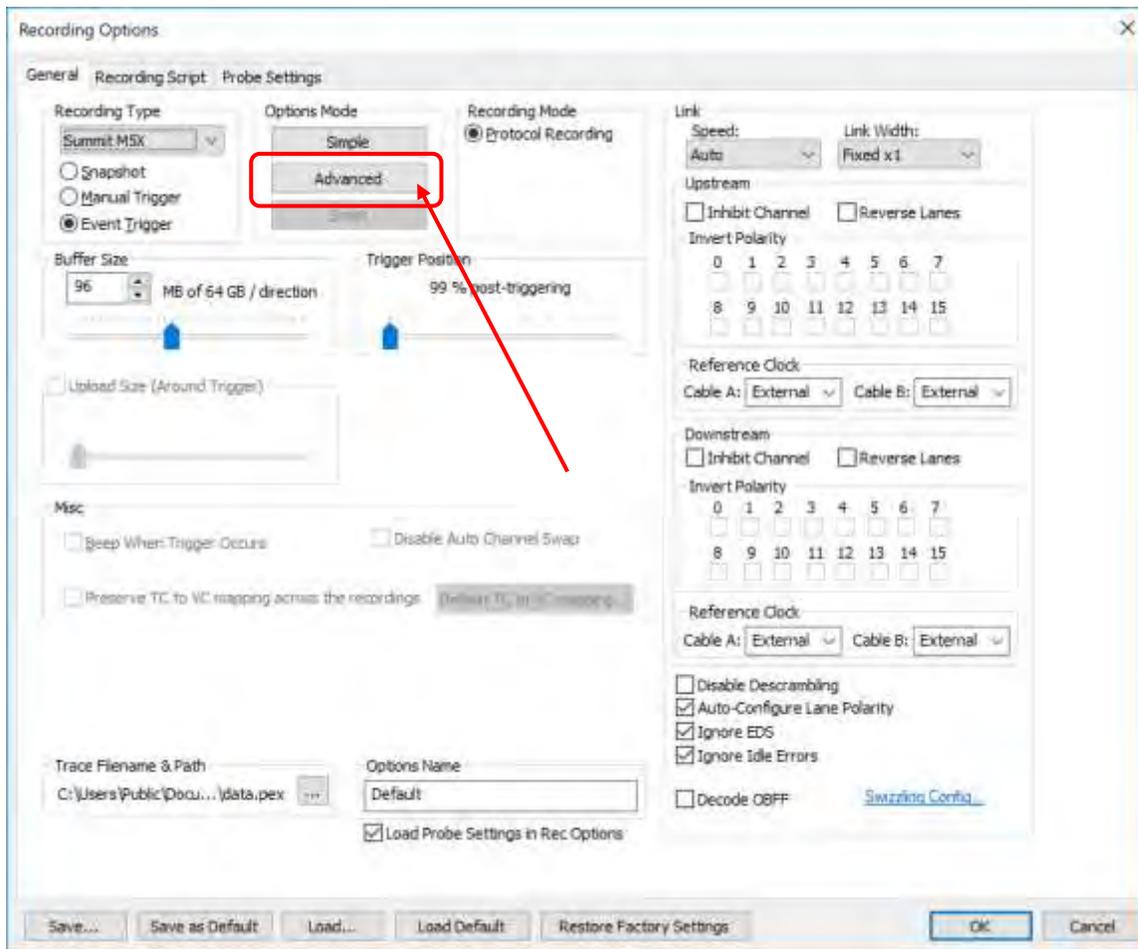
and the Jammer Actions file will be available. See below.



Select the link\_layer\_Jammer\_Actions.rec file and click on "Open". The file will be loaded as a script. See below.

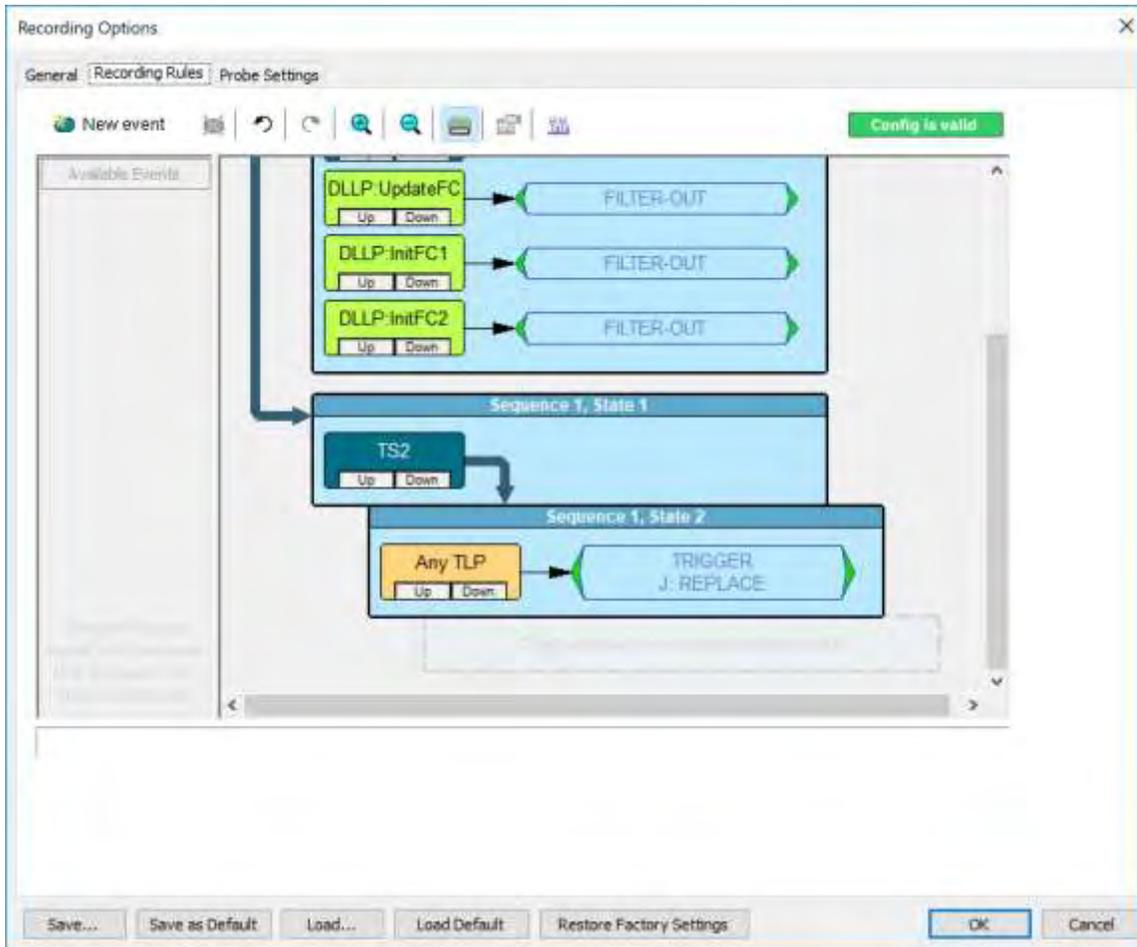


If you want to see it in the GUI format, select General -> Advanced (see below)



Select Advanced and the Recording Script tab will change to Recording Rules.

Select the Recording Rules tab and the GUI for the Jammer Actions file will pop up. See below.



## 3.8 Examples of Scripting for Events with Jammer Actions

The following are examples common events and the script syntax for Jammer Actions:

Event	Event Type
Link State	Speed Change
Ordered Set	TS2
Framing Token	EDS
Errors	Symbol Error
DLLP	Any
TLP	Any
NVMe Register	ACQ
NVMe Command Submission	Any
NVMe Command Completion	Any

There are seven types of Jammer Actions which can be associated with any of the Events shown above:

- None
- Delete
- Replace
- Modify
- Insert Pkt X after
- Insert Error
- Sideband Signal

### 3.8.1 Link State: Speed Change

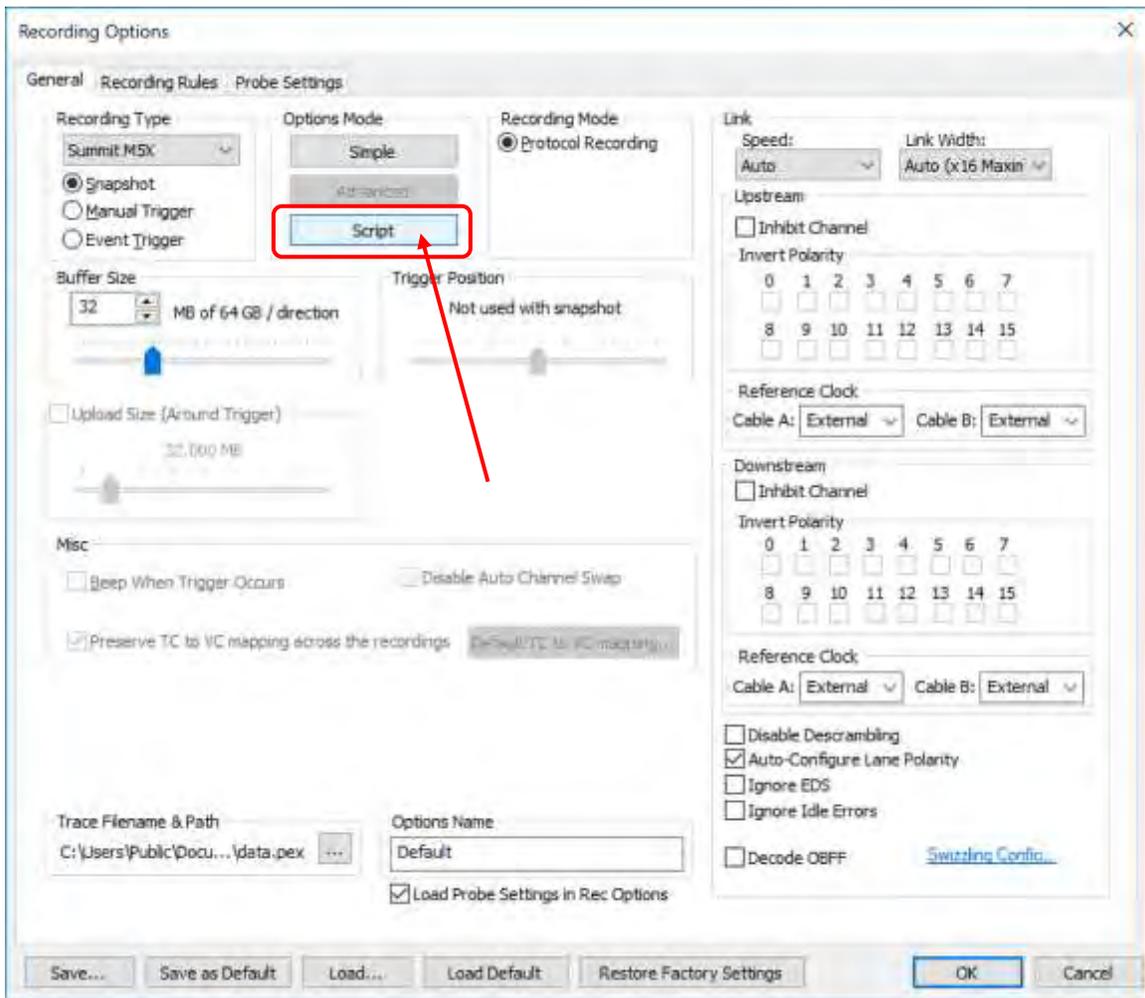
Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

Then select New Event -> Link State -> Speed Change -> Insert Pkt X after

See GUI dialog below.



Select the General tab to see the generated script.



Now select Script, then click on the Recording Script tab.

See the resulting script below.

Recording Options

General Recording Script Probe Settings

```

1 RecAction = Jammer
2 {
3     ActionName = "JammerAction_0"
4     Count = 7
5     JammerType = JammerInsertPktXAfter
6     PacketType = Packet_L_IDLE
7     EDSInsDelBeforeOS = Yes
8     AlignNextLane0Pkt = Yes
9     Packet_Length = 1
10    G1G2_Raw_Data = (
11        "0x00000000"
12    )
13    G1G2_KBits_Data = (
14        "0b0000"
15    )
16    G3G4_Raw_Data = (
17        "0x00000000"
18    )
19 }
20 RecEvent = LinkState
21 {
22     Channels = All
23     CurrentState = State_Global
24     Action = ( "JammerAction_0" )
25     EnterEI = No
26     ExitEI = No
27     CLKREQ_Asserted = No
28     CLKREQ_Deasserted = No
29     WAKE_Asserted = No
30     WAKE_Deasserted = No
31     PERST_Asserted = No
32     PERST_Deasserted = No
33     SpeedSwitch_From_G1 = Yes
34     SpeedSwitch_From_G2 = Yes
35     SpeedSwitch_From_G3 = Yes
36     SpeedSwitch_From_G4 = Yes

```

Parameter	Value
ActionName	"JammerAction_0"
Count	7
JammerType	JammerInsertPktXAfter
PacketType	Packet_L_IDLE
EDSInsDelBeforeOS	Yes
AlignNextLane0Pkt	Yes
Packet_Length	1
G1G2_Raw_Data	("0x00000000")
G1G2_KBits_Data	("0b0000")
G3G4_Raw_Data	("0x00000000")

Line Description

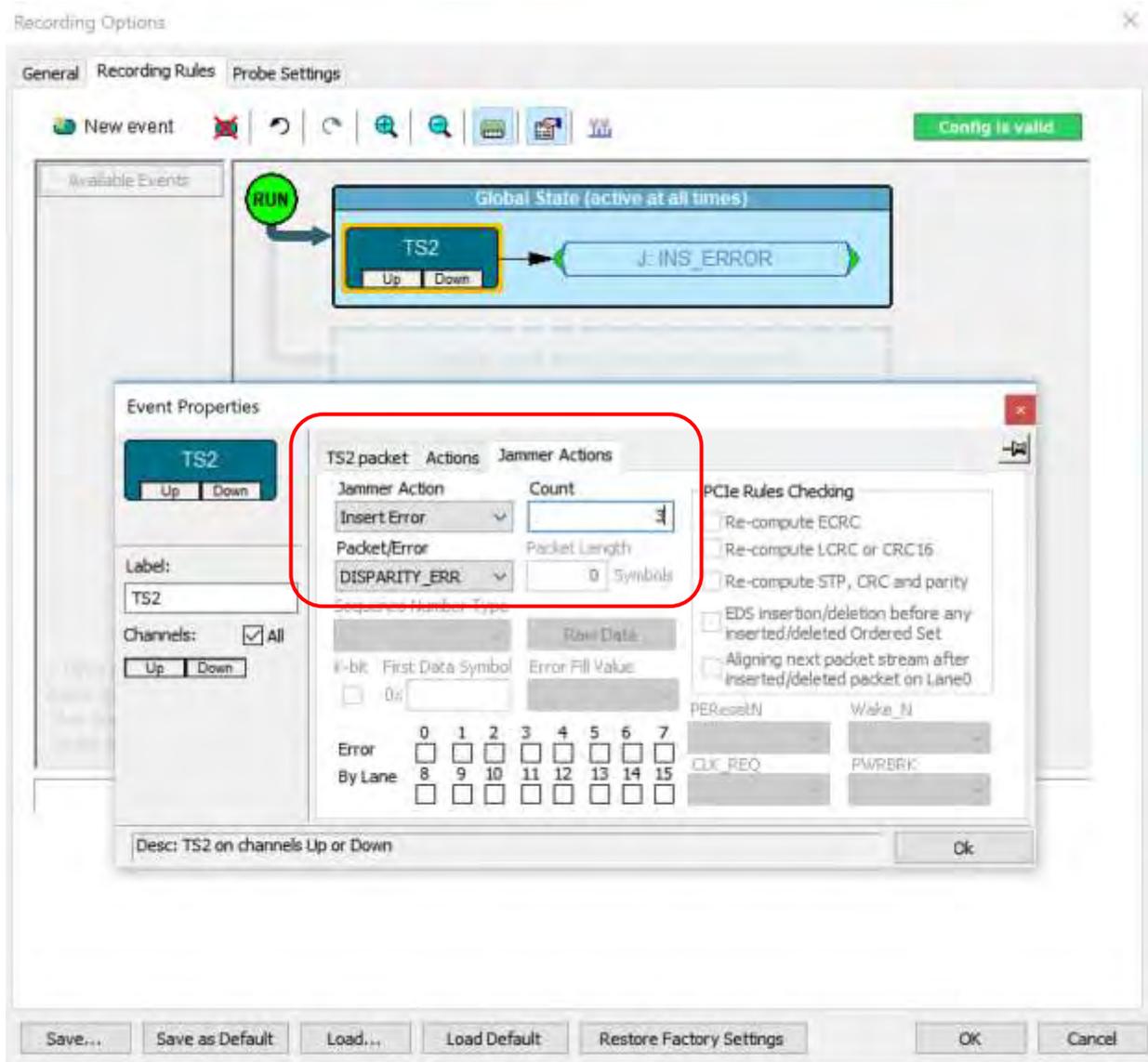
Save... Save as Default Load... Load Default Restore Factory Settings OK Cancel

### 3.8.2 Ordered Set: TS2

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> Ordered Set -> TS2 -> Insert Error

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with the 'Recording Script' tab selected. The script editor contains the following code:

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 3
5   JammerType = JammerInsertError
6   ErrorType = DISPARITY_ERR
7 }
8
9 RecEvent = TS2
10 {
11   Channels = All
12   CurrentState = State_Global
13   Action = ( "JammerAction_0" )
14 }
```

Two red boxes highlight the script and a parameter table. The first box encloses the script code for the 'RecAction' block. The second box encloses the parameter table on the right, which lists the values for the script parameters. A red arrow points from the 'ActionName' parameter in the table to its corresponding value in the script.

Parameter	Value
ActionName	"JammerAction_0"
Count	3
JammerType	JammerInsertError
ErrorType	DISPARITY_ERR
ErrorByLane	

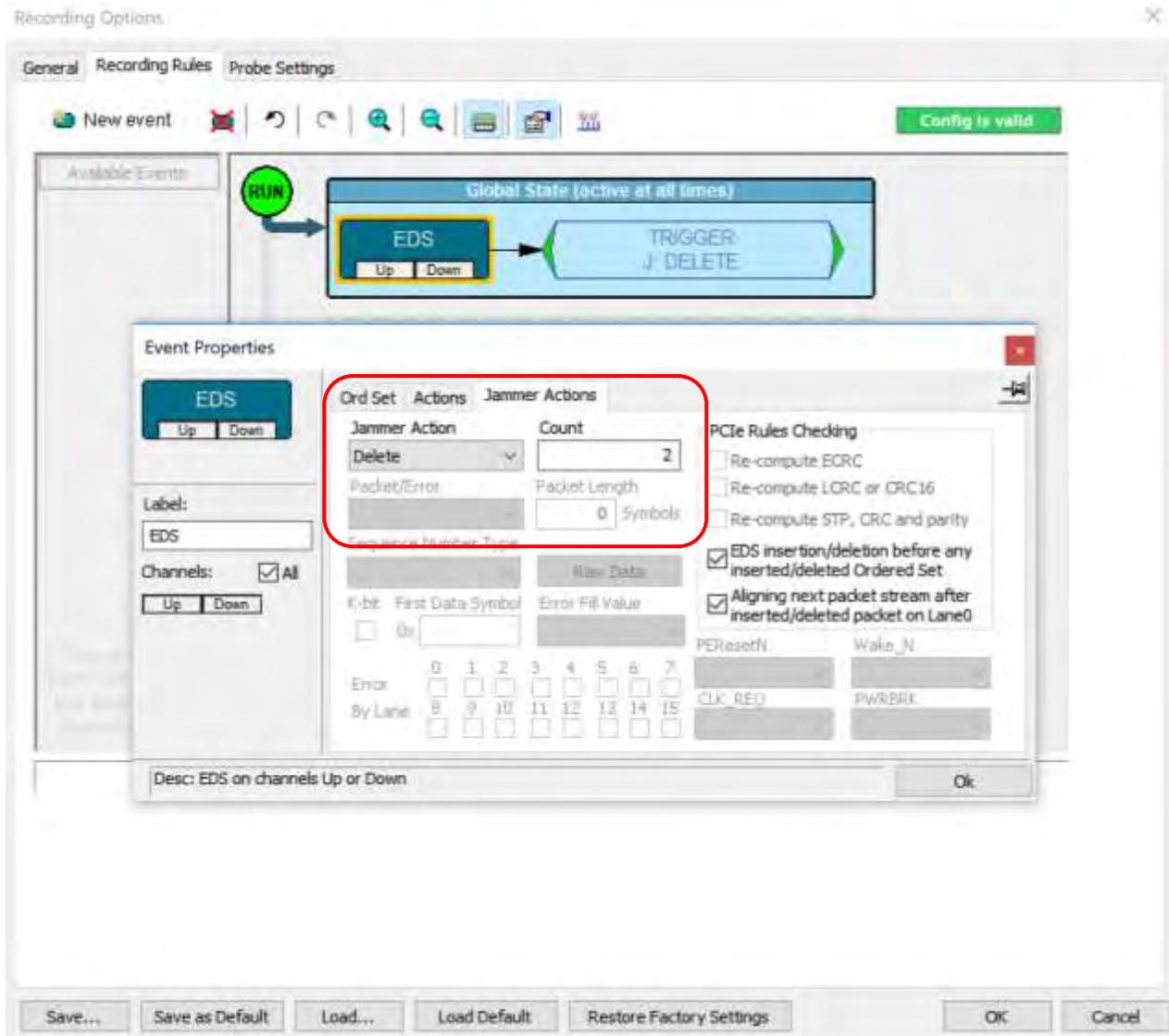
At the bottom of the dialog, there is a table with two columns: 'Line' and 'Description'. Below this table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

### 3.8.3 Framing Token: EDS

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> Framing Token -> EDS -> Delete

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with the 'Recording Script' tab selected. The script defines a 'RecAction' named 'Jammer' and a 'RecEvent' named 'OrderedSet'. The 'RecAction' block contains the following parameters:

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 2
5   JammerType = JammerDelete
6   EDSInsDelBeforeOS = Yes
7   AlignNextLaneOPkt = Yes
8 }
9
10 RecEvent = OrderedSet
11 {
12   Channels = All
13   CurrentState = State_Global
14   Action = ( Trigger, "JammerAction_0" )
15   SKP = No
16   CTRL_SKP = No
17   EIOS = No
18   EDG = Yes
19   EIEOS = No
20   FTS = No
21   SDS = No
22   EDB = No
23 }
```

Red boxes highlight the 'RecAction' block in the script and a table of its parameters. An arrow points from the 'ActionName' parameter in the script to the table. The table lists the following parameters and values:

Parameter	Value
ActionName	"JammerAction_..."
Count	2
JammerType	JammerDelete
EDSInsDelBefor...	Yes
AlignNextLane0...	Yes

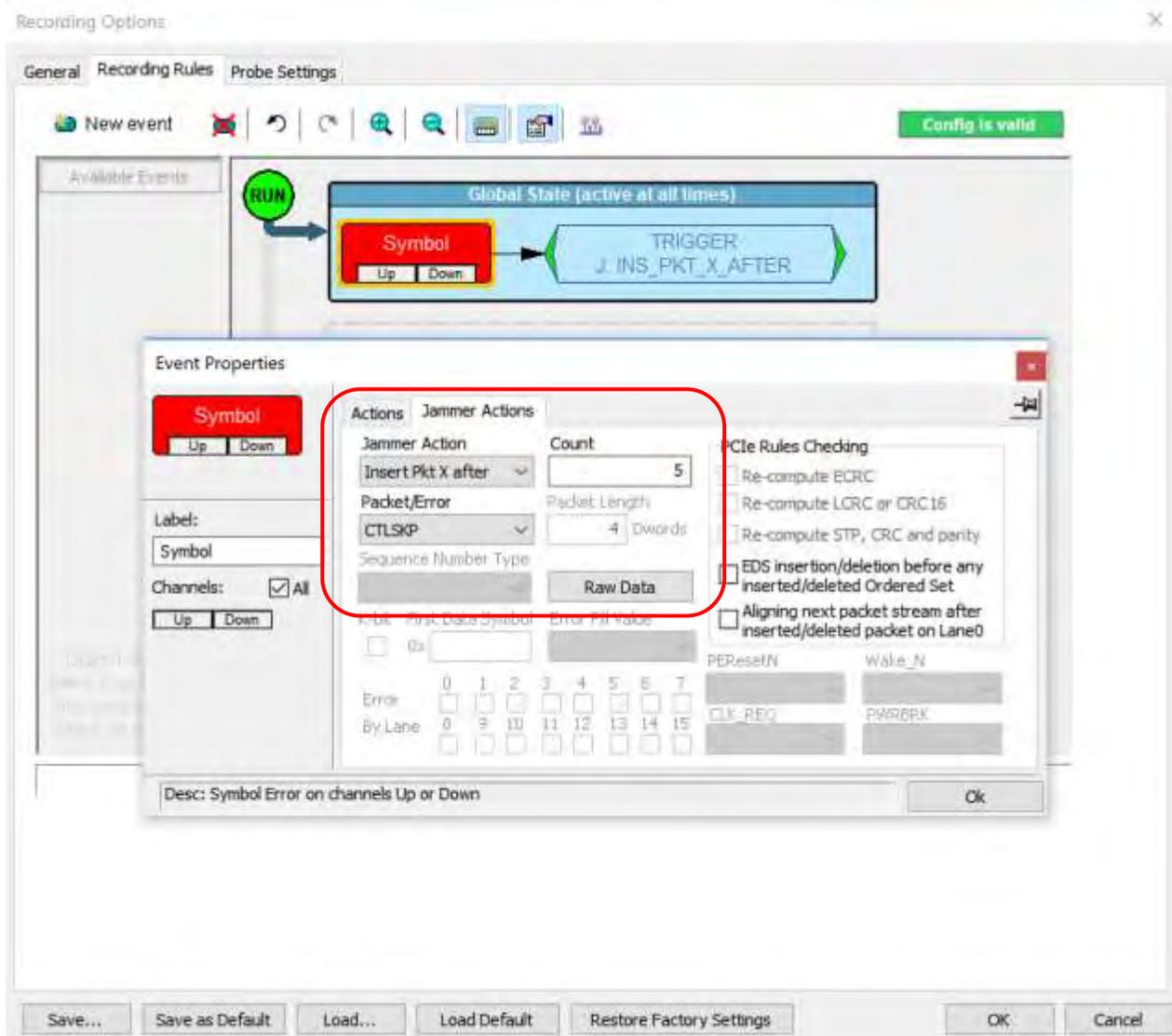
At the bottom of the dialog, there is a table with two columns: 'Line' and 'Description'. The 'Line' column is empty, and the 'Description' column contains the text 'Description'. Below the table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

### 3.8.4 Errors: Symbol Error

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> Errors -> Symbol Error-> Replace

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with three tabs: 'General', 'Recording Script', and 'Probe Settings'. The 'Recording Script' tab is active, displaying a script for a 'Jammer' action. A red box highlights the script code, and another red box highlights the parameter table on the right.

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 5
5   JammerType = JammerInsertPktXAfter
6   PacketType = Packet_CTRLSKP
7   G3G4_Raw_Data = (
8     "0xaaaaaaaa", "0xaaaaaaaa", "0xaaaaaaaa",
9   )
10 }
11 RecEvent = SymbolError
12 {
13   Channels = All
14   CurrentState = State_Global
15   Action = ( Trigger, "JammerAction_0" )
16 }
17 }
```

Parameter	Value
ActionName	"JammerAction_0"
Count	5
JammerType	JammerInsertPktXAfter
PacketType	Packet_CTRLSKP
EDSinsDelBeforeOS	
AlignNextLaneOPkt	
G3G4_Raw_Data	("0xaaaaaaaa", "0xaaa...

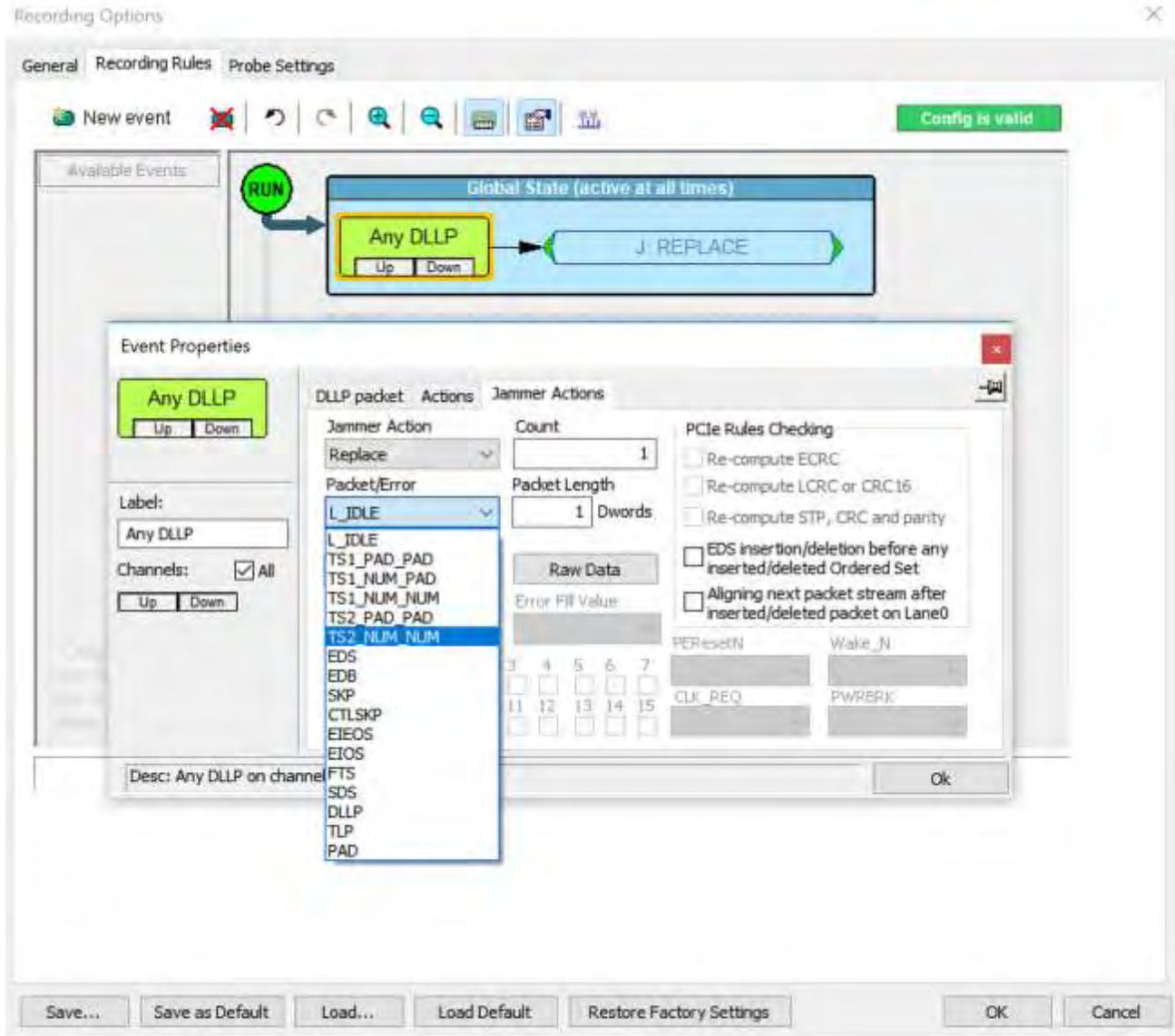
At the bottom of the dialog, there is a table with columns 'Line' and 'Description', which is currently empty. Below the table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

### 3.8.5 DLLP: Any

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> DLLP -> Any -> Replace (TS2\_NUM\_NUM)

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with two tabs: 'General' and 'Recording Script'. The 'Recording Script' tab is active, displaying a script for a jammer action. The script defines a 'RecAction' named 'Jammer' with various parameters like 'ActionName', 'Count', 'JammerType', and raw data for different protocols. A table on the right lists these parameters and their values. The bottom of the dialog has a 'Line' and 'Description' table, and buttons for 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 1
5   JammerType = JammerReplace
6   PacketType = Packet_TS2_NUM_NUM
7   G1G2_Raw_Data = (
8     "0xbc000000", "0x00004545", "0x45454545"
9   )
10  G1G2_KBits_Data = (
11    "0b1000", "0b0000", "0b0000", "0b0000"
12  )
13  G3G4_Raw_Data = (
14    "0x2d000000", "0x00000045", "0x45454545"
15  )
16 }
17 RecEvent = DLLP
18 {
19   Channels = All
20   CurrentState = State_Global
21   Action = ( "JammerAction_0" )
22 }
23
```

Parameter	Value
ActionName	"JammerAction_0"
Count	1
JammerType	JammerReplace
PacketType	Packet_TS2_NUM_NUM
EDSinsDelBeforeOS	
AlignNextLaneOPkt	
G1G2_Raw_Data	("0xbc000000", "0x000045...
G1G2_KBits_Data	("0b1000", "0b0000", "0b0...
G3G4_Raw_Data	("0x2d000000", "0x000000...

Line	Description
------	-------------

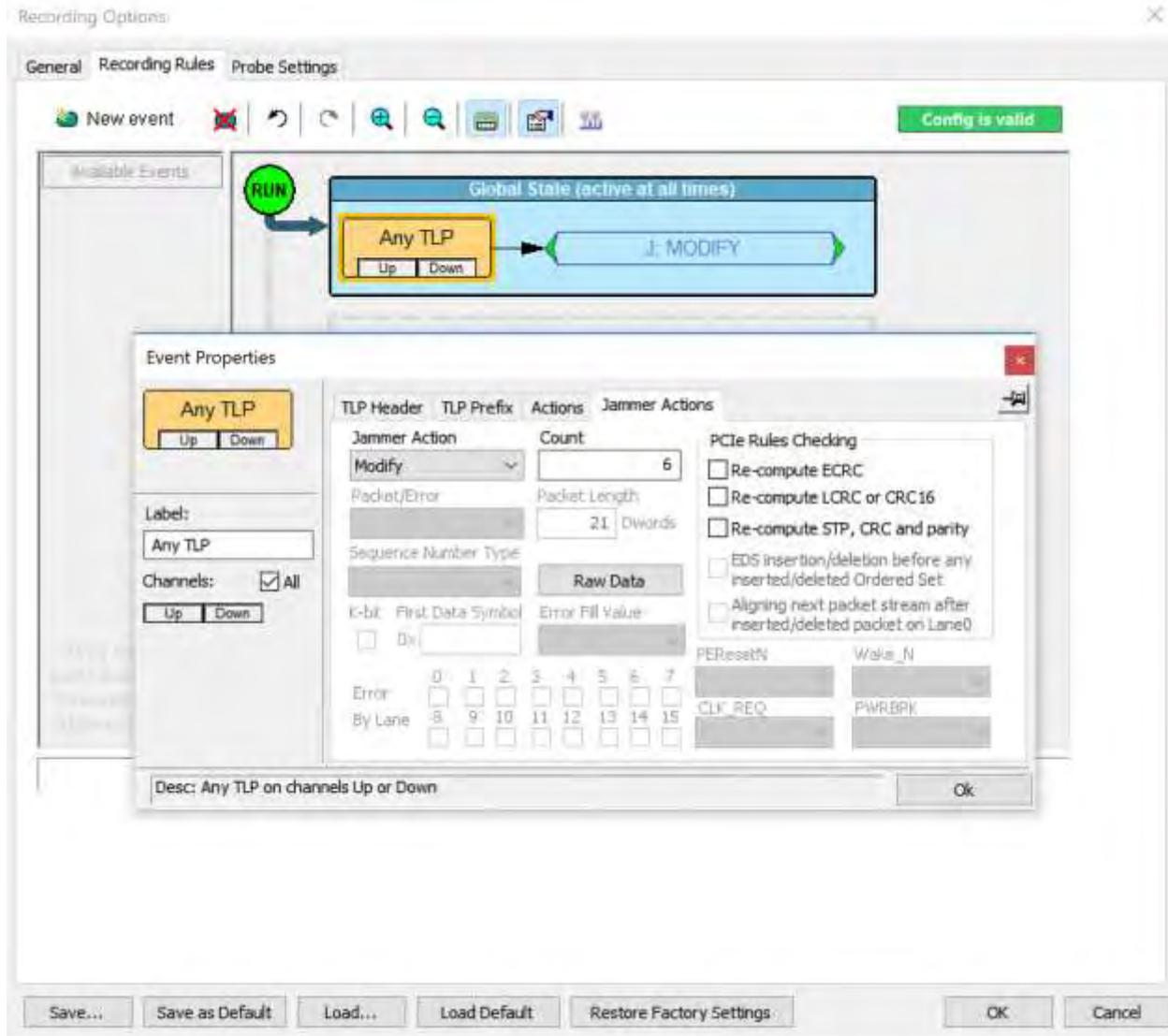
Save... Save as Default Load... Load Default Restore Factory Settings OK Cancel

### 3.8.6 TLP: Any

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> TLP -> Any -> Modify

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with three tabs: 'General', 'Recording Script', and 'Probe Settings'. The 'Recording Script' tab is active, displaying a script with two main sections: 'RecAction = Jammer' and 'RecEvent = TLP'. The 'RecAction' section is highlighted with a red box and contains the following code:

```
RecAction = Jammer
{
  ActionName = "JammerAction_0"
  Count = 6
  JammerType = JammerModify
  SourceEventType = TLP
}
```

The 'RecEvent' section is also highlighted with a red box and contains the following code:

```
RecEvent = TLP
{
  Channels = All
  CurrentState = State_Global
  Action = ( "JammerAction_0" )
  TLP_Prefix = No
}
```

Below the script editor is a table with two columns: 'Parameter' and 'Value'. The table lists various parameters and their corresponding values, with the first four rows highlighted by a red box:

Parameter	Value
ActionName	"JammerAction_0"
Count	6
JammerType	JammerModify
SourceEventType	TLP
Recompute_TlpECRC	
Recompute_TlpLCRC_DllpCR...	
Recompute_TlpSTPTokenCRC	
G1G2_Modify_TLP_Prefix	
G3G4_Modify_TLP_Prefix	
G1G2_Raw_Data	
G3G4_Raw_Data	

At the bottom of the dialog, there is a table with two columns: 'Line' and 'Description', which is currently empty. Below this table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

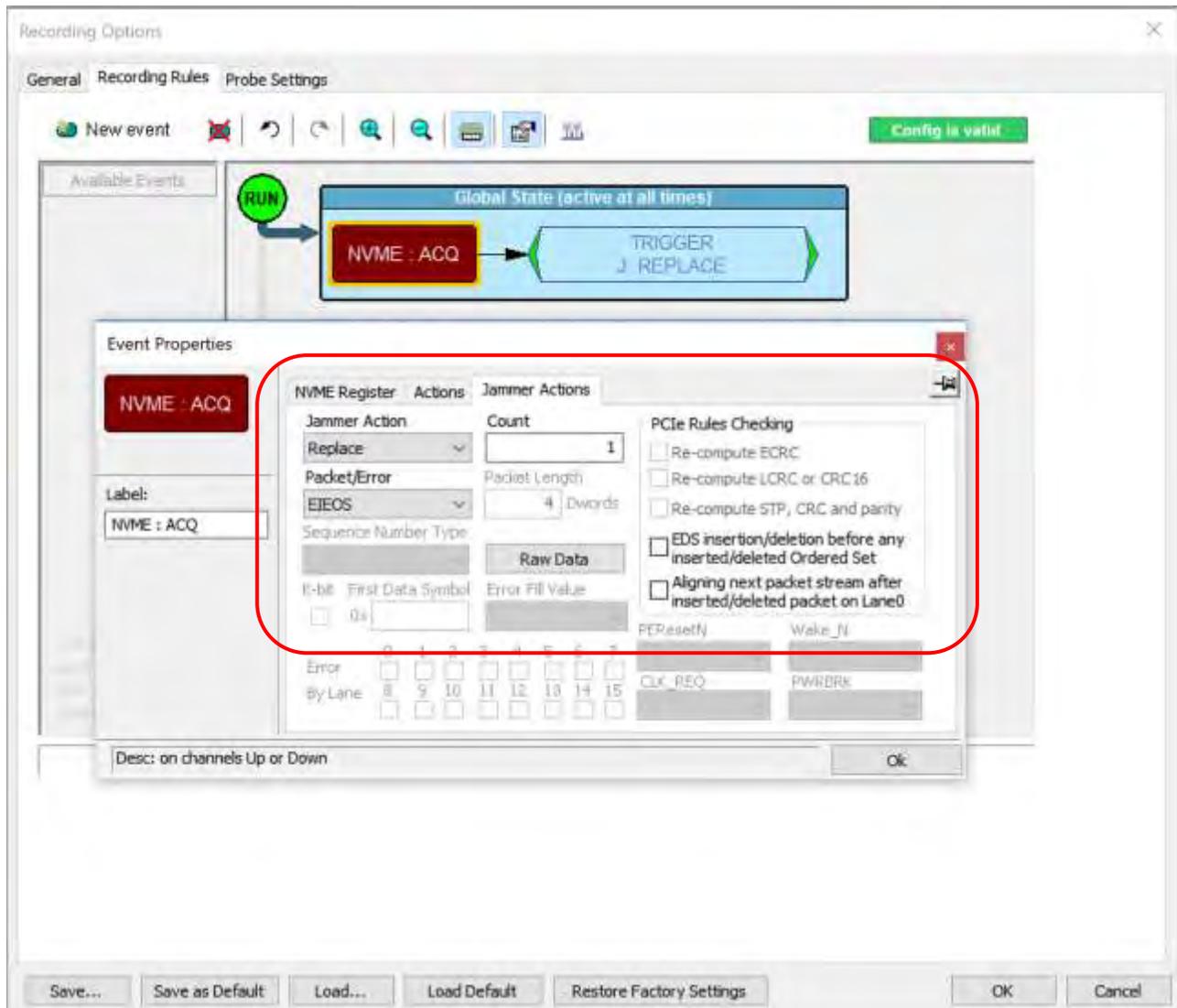
## 3.8.7 NVMe Events

### 3.8.7.1 NVMe Register

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> NVMe Register -> ACQ -> Replace

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with three tabs: 'General', 'Recording Script', and 'Probe Settings'. The 'Recording Script' tab is active, displaying a script with two main sections: 'RecAction = Jammer' and 'RecEvent = NYMe\_Register'. A red box highlights the 'RecAction' block, and another red box highlights the parameter table on the right. A red arrow points from the 'RecAction' block to the parameter table.

```
1 RecAction = Jammer
2 {
3     ActionName = "JammerAction_0"
4     Count = 1
5     JammerType = JammerReplace
6     PacketType = Packet_EIEOS
7 }
8 RecEvent = NYMe_Register
9 {
10    CurrentState = State_Global
11    Action = ( Trigger, "JammerAction_0" )
12    BAR_0 = 0x0
13    BAR_1 = 0x0
14    Type = ACQ
15 }
16
```

Parameter	Value
ActionName	"JammerAction_0"
Count	1
JammerType	JammerReplace
Packet Type	Packet_EIEOS
EDSInsDelBeforeOS	
AlignNextLane0Pkt	
G1G2_Raw_Data	
G1G2_KBits_Data	
G3G4_Raw_Data	

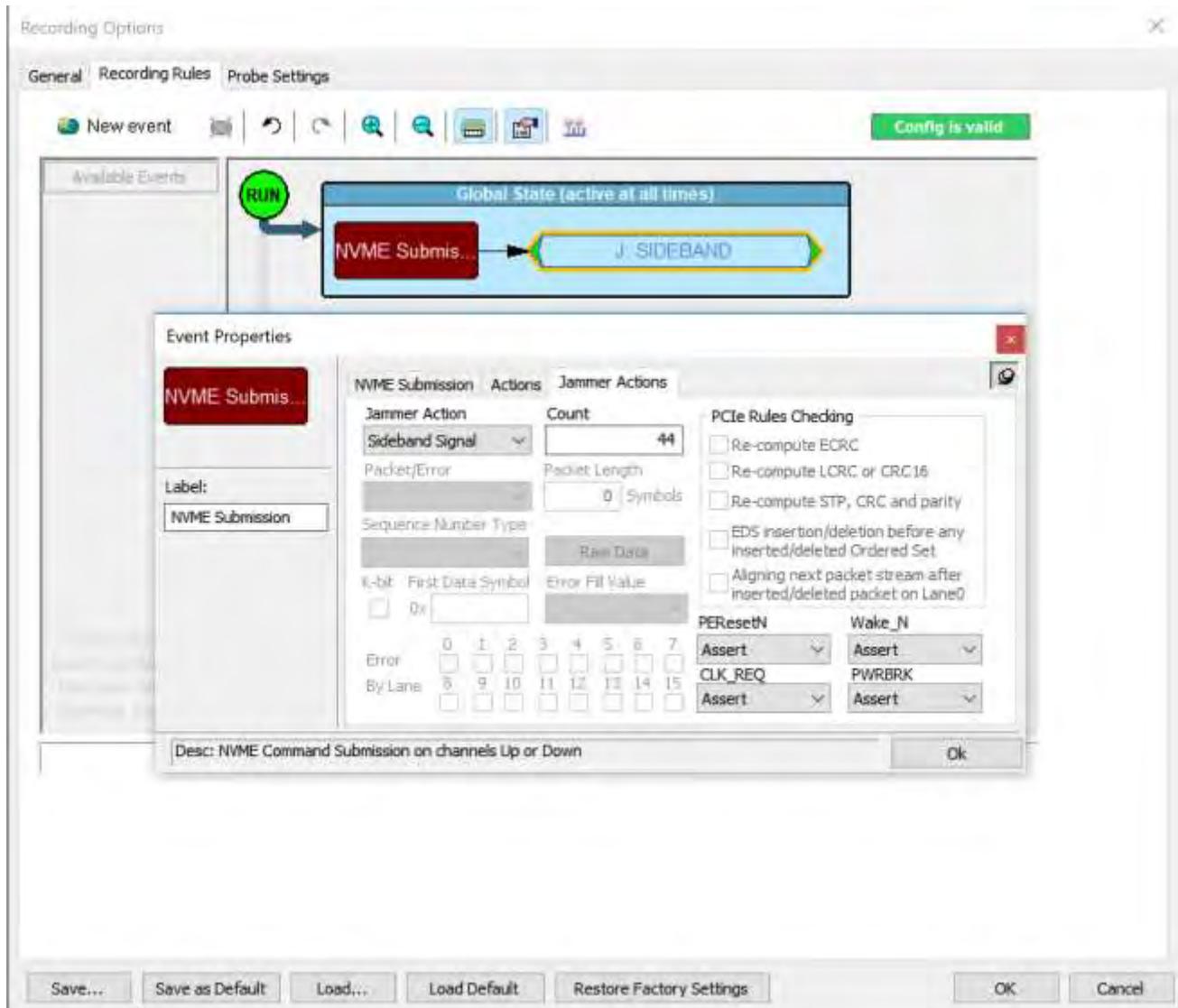
At the bottom of the dialog, there is a table with two columns: 'Line' and 'Description'. Below the table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

### 3.8.7.2 NVMe Command Submission: Any

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> NVMe Command Submission -> Any -> Sideband Signal

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with three tabs: 'General', 'Recording Script', and 'Probe Settings'. The 'Recording Script' tab is active, displaying a script with two main sections: 'RecAction = Jammer' and 'RecEvent = NVMe\_Command\_Submission'. The 'RecAction' section is highlighted with a red box and contains the following code:

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 44
5   JammerType = JammerSidebandSignal
6   PEResetN = SB_Assert
7   CLK_REQ = SB_Assert
8   WAKE_N = SB_Assert
9   PWRBRK = SB_Assert
10 }
11 RecEvent = NVMe_Command_Submission
12 {
13   CurrentState = State_Global
14   Action = ( "JammerAction_0" )
15   FUSE = Normal_Operation
16 }
17
```

The 'RecEvent' section is also highlighted with a red box and contains the following code:

```
12 {
13   CurrentState = State_Global
14   Action = ( "JammerAction_0" )
15   FUSE = Normal_Operation
16 }
```

To the right of the script is a table of parameters and their values, also highlighted with a red box:

Parameter	Value
ActionName	"JammerAction_0"
Count	44
JammerType	JammerSidebandSignal
PEResetN	SB_Assert
CLK_REQ	SB_Assert
WAKE_N	SB_Assert
PWRBRK	SB_Assert

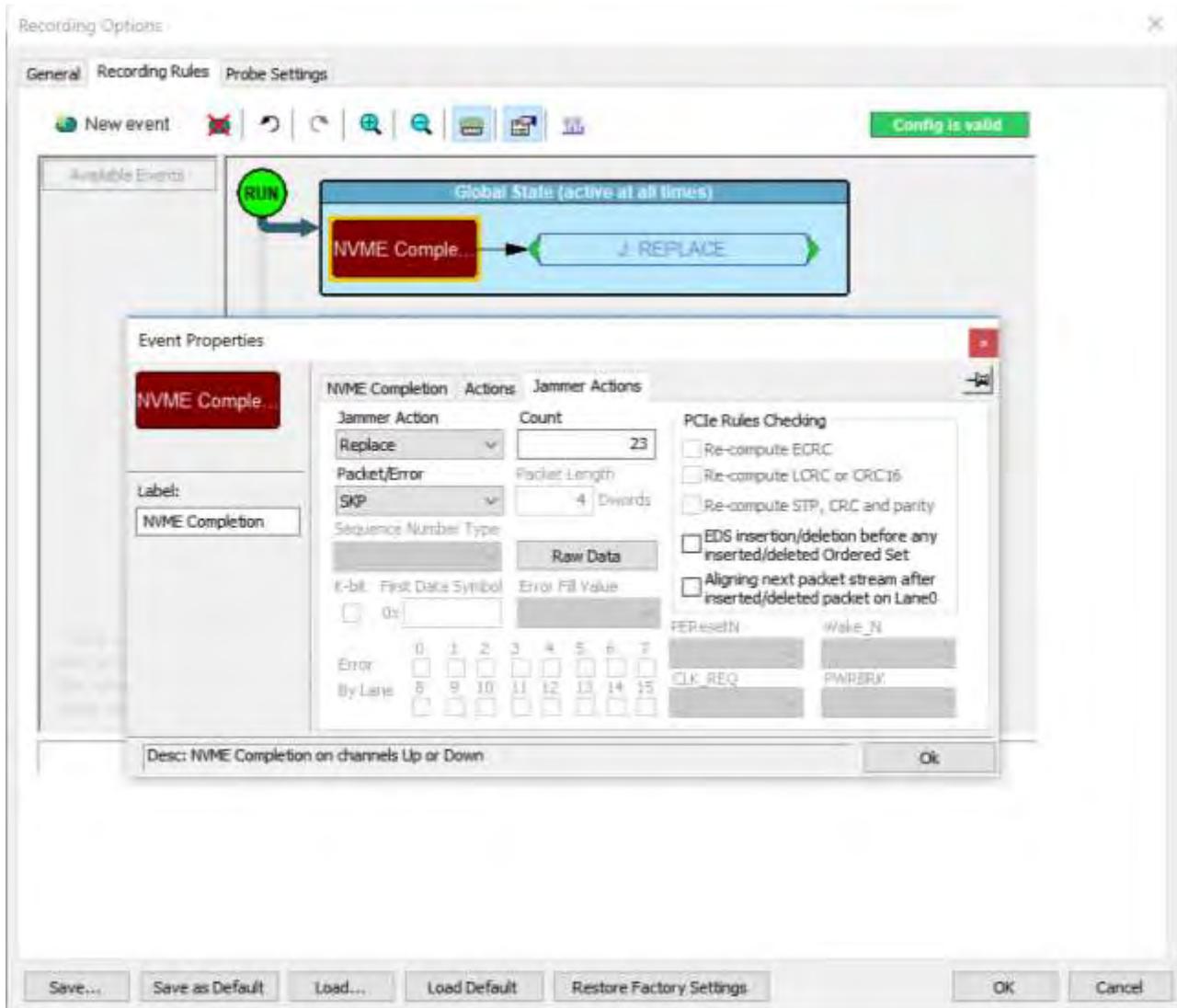
At the bottom of the dialog is a table with two columns: 'Line' and 'Description'. Below the table are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

### 3.8.7.3 NVMe Command Completion: Any

Set up the PCIe Protocol Analysis software for the M5x -> Set -> Recording Options -> Recording Rules.

The select New Event -> NVMe Command Completion -> Any -> Replace

See GUI dialog below.



Select the General tab to see the generated script.

Now select Script, then click on the Recording Script tab.

See the resulting script below.

The screenshot shows the 'Recording Options' dialog box with three tabs: 'General', 'Recording Script', and 'Probe Settings'. The 'Recording Script' tab is active, displaying a script with the following code:

```
1 RecAction = Jammer
2 {
3   ActionName = "JammerAction_0"
4   Count = 23
5   JammerType = JammerReplace
6   PacketType = Packet_SKP
7   G1G2_Raw_Data = (
8     "0xbc1c1c1c"
9   )
10  G1G2_KBits_Data = (
11    "0b1111"
12  )
13  G3G4_Raw_Data = (
14    "0xaaaaaaaa", "0xaaaaaaaa", "0xaaaaaaaa",
15  )
16 }
17 RecEvent = NVMe_Command_Completion
18 {
19   CurrentState = State_Global
20   Action = ( "JammerAction_0" )
21 }
22 }
```

To the right of the script is a table with two columns: 'Parameter' and 'Value'. The table contains the following entries:

Parameter	Value
ActionName	"JammerAction_0"
Count	23
JammerType	JammerReplace
PacketType	Packet_SKP
EDSinsDelBeforeOS	
AlignNextLane0Pkt	
G1G2_Raw_Data	("0xbc1c1c1c")
G1G2_KBits_Data	("0b1111")
G3G4_Raw_Data	("0xaaaaaaaa", "0xaaaaaa...

Below the script and table is a table with two columns: 'Line' and 'Description'. This table is currently empty.

At the bottom of the dialog box are several buttons: 'Save...', 'Save as Default', 'Load...', 'Load Default', 'Restore Factory Settings', 'OK', and 'Cancel'.

## 4 Detailed Description of Script Mode Syntax

### 4.1 General Syntax Rules

Summit M5X™ Script Mode in recording rules 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 58. 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 notations are supported:

- Hexadecimal numbers must be preceded by 0x: 0x2A, 0x54, 0xFFFFFFFF01
- Decimal numbers are written as usual: 24, 1256, 2
- Binary numbers are denoted with 0b: 0b01101100, 0b01, 0b100000

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.

## 4.2 Command List

COMMAND	MODIFIERS	Comment
<a href="#">RecEvent</a>	TLP DLLP OrderedSet Skip CtrlSkip TS1 TS2 LinkState Error Lane_Margining NVMe_Command_Completion NVMe_Command_Submission NVMe_Register MCTP SMBus_ARP Global_Timer_A Global_Timer_B Local_Timer_A Local_Timer_B Local_Timer_C Local_Timer_D Global_Counter_1 Global_Counter_2 Local_Counter_1 Local_Counter_2 Local_Counter_3 Local_Counter_4	Creates specific event in recording rules.
<a href="#">RecAction</a>	Jammer	Creates jammer action with complex settings, that can be assigned to event

## 4.3 RecEvent Command

This command creates specified event in recording rules.

### 4.3.1 Common for all RecEvent fields

RecEvent has the following properties, which are common for all types of events:

Parameter	Value	Default	Comment
Channels	All, Upstream, Downstream, None	All	Channels of event in recording rules: <b>All</b> – event on upstream (from device to host) and downstream (from host to device) traffic, <b>Upstream</b> – event on upstream traffic only (from device to host), <b>Downstream</b> – on downstream traffic only (from host to device), <b>None</b> – no channel specified.
CurrentState	State_Global, State_Repository, State_1, State_2, State_3, State_4, State_5, State_6, State_7, State_8, State_9, State_10, State_11, State_12, State_13, State_14, State_15, State_16	State_Global	State of event in recording rules sequencer. <b>State_Global</b> – event is always active in any state of sequencer. <b>State_Repository</b> – event is not active in any state yet, doesn't take part in recording. <b>State_1, ..., State_16</b> – event is active and takes place in one of sixteen local states of sequencer.
Action	Trigger, Filter_In, Filter_Out, Start_Global_Timer_A, Reset_Global_Timer_A, Start_Global_Timer_B, Reset_Global_Timer_B, Increment_Global_Counter_1, Reset_Global_Counter_1, Increment_Global_Counter_2, Reset_Global_Counter_2, Start_Local_Timer_A, Reset_Local_Timer_A,		Action that is related to event in recording rules: <b>Trigger</b> – event will cause analyzer to trigger during recording, <b>Filter_In</b> – event will be filtered in and will be present in recorded trace, <b>Filter_Out</b> – event will be filtered out and will not be present in recorded trace, <b>Start_Global_Timer_A/B</b> – event will start one of global timers (A/B) <b>Reset_Global_Timer_A/B</b> – event will reset one of global timers (A/B)

	<p>Start_Local_Timer_B,  Reset_Local_Timer_B,  Start_Local_Timer_C,  Reset_Local_Timer_C,  Start_Local_Timer_D,  Reset_Local_Timer_D,  Increment_Local_Counter_1,  Reset_Local_Counter_1,  Increment_Local_Counter_2,  Reset_Local_Counter_2,  Increment_Local_Counter_3,  Reset_Local_Counter_3,  Increment_Local_Counter_4,  Reset_Local_Counter_4,  Advance_Sequence,  Reset_Sequence,  External_Trigger_High,  External_Trigger_Low,  External_Trigger_Toggle</p>		<p><b>Increment_Global_Counter_1/2-</b>  event will increment one of global counters (1/2)  <b>Reset_Global_Counter_1/2</b> – event will reset one of global counters (1/2)  <b>Start_Local_Timer_A/B/C/D</b> – event will start one of local timers (A/B/C/D)  <b>Reset_Local_Timer_A/B/C/D</b> – event will reset one of local timers (A/B/C/D)  <b>Increment_Local_Counter_1/2/3/4</b> – event will increment one of local counters (1/2/3/4)  <b>Reset_Local_Counter_1/2/3/4</b> – event will reset one of local counters (1/2/3/4)  <b>Advance_Sequence</b> – event will advance the sequence (sequencer moves to another state)  <b>Reset_Sequence</b> – event will reset the sequence  <b>External_Trigger_High</b> - event will set external signal to high state  <b>External_Trigger_Low</b> – event will set external signal to low state  <b>External_Trigger_Toggle</b> – event will toggle external signal state</p>
TargetStateNumber	<p>State_1,  State_2,  State_3,  State_4,  State_5,  State_6,  State_7,  State_8,  State_9,  State_10,  State_11,  State_12,  State_13,  State_14,  State_15,  State_16</p>		<p>This parameter allows selecting next state for Advance_Sequence action. It can be specified if event is located in one of sixteen local states and has Advance_Sequence action.</p>

### 4.3.2 **RecEvent = TLP**

This command creates TLP event in recording rules. The parameters of the **RecEvent = TLP** command cover all the fields in the TLP header.

### 4.3.2.1 Common for all types of TLP's fields

Parameter	Values	Default Value	Comment
TLPType	Mem IO Cfg TCfg Msg Cpl FetchAdd Swap CAS		Sets the <b>Fmt</b> (bits 6:5 of byte 0 in the TLP header) and <b>Type</b> (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 specifies bits 6:0 of byte 0 in the TLP header.
TC	0:7		<b>Traffic Class:</b> bits 6:4 of byte 1 in the TLP header
Digest	Present Not_Present		Bit 7 of byte 2 in the TLP header: Indicates presence of TLP digest in the form of a single DW at the end of the TLP.
Poisoned	Yes No		Bit 6 of byte 2 in the TLP header: Indicates if the TLP is poisoned.
Snoop	Yes No		Bit 4 of byte 2 in the TLP header: "No" indicates that hardware enforced cache coherency is expected. "Yes" indicates that hardware enforced cache coherency is not expected.
Fmt	3DW_no_data 3DW_with_data 4DW_no_data 4DW_with_data		Corresponding TLP Format: 3DW_no_data - 3 DW header, no data 3DW_with_data - 3 DW header, with data 4DW_no_data - 4 DW header, no data 4DW_with_data - 4 DW header, with data
Ordering	Strong Relaxed		Bit 5 of byte 2 of TLP header: "Strong" indicates PCI Strongly Ordered Model. "Relaxed" indicates PCI-X Relaxed Ordering Model.
Address Type	0-3		Address Type: Bits 2-3 of Byte 2 in the TLP header.
Length	0:1023		Length of data payload in DWORDs. For a length of 1024, set Length to 0 (because 0 means 1024)
Tag_Bit_8	0-1		Tag bit 8 for 10 bits Tag. Bit 3 of Byte 1 of the TLP Header
Tag_Bit_9	0-1		Tag Bit 9 for 10 bits Tag. Bit 7 of Byte 1 of the TLP Header
Hints	Present Not_Present		Bit 0 of Byte 1 of the TLP Header. Indicates the presence of TLP Processing Hints
ID_Based_Ordering	Yes No		Indicates independent ordering based on Requester/Completer ID

Parameter	Values	Default Value	Comment
TLP_Data	(XXXX,XXXX,...)		Specified as the array of DWORDs in hexadecimal format (Big Endian). This parameter applies only to TLP packets with data.
TLP_Prefix	Yes No		Indicates if TLP prefix is present

Example:

Create TLP Mem event in recording rules with Length 21 DW and action Trigger

```
RecEvent = TLP {  
    TLPType = Mem  
    Length = 21  
    Action = Trigger  
}
```

### 4.3.2.2 TLPType = Mem (32 bit and 64 bit)

Parameter	Value	Default	Comment
BE_First	0:15		Byte 7 in the TLP header. See rules for <b>Last DW BE</b> in the PCI Express Specification.
BE_Last	0:15		Byte 7 in the TLP header. See rules for <b>1st DW BE</b> in the PCI Express Specification.
Address (for 32-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 8-11 in the TLP header in case of 32-bit TLP.
Address_HI (for 64-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 8-11 in the TLP header in case of 64-bit TLP.
Address_LO (for 64-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 12-15 in the TLP header in case of 64-bit TLP.
Requester_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Requester ID in the TLP.
Requester_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Requester ID in the TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of memory TLP.

#### Example 1:

This example shows how to create a 32-bit Memory Write TLP event in recording rules with trigger action.

```

Packet = TLP {
    Channels = All
    CurrentState = State_Global
    Action = ( Trigger )
    TLPType = Mem
    Fmt = "0bX1X"
    BE_Last = 0xF
    BE_First = 0xF
    Address = 0x1000
    TLP_Data = ( 0x2, 0x4, 0x6, 0x8 )
}

```

#### Example 2:

This example shows how to create a 64-bit Memory Write TLP event in recording rules.

```

Packet = TLP {
    TLPType = Mem
    Fmt = 4DW_with_data
    BE_First = 0xF
    BE_Last = 0xF
    Address_LO = 0x1000
    Address_HI = 0x60000000
    TLP_Data = ( 0x2, 0x4, 0x6, 0x8, 0x2, 0x4, 0x6, 0x8 )
}

```

```
}
```

### 4.3.2.3 TLPType = IO

Parameter	Value	Default	Comment
BE_First	0:15		Byte 7 in the TLP header. See rules for <b>Last DW BE</b> in the PCI Express Specification.
BE_Last	0:15		Byte 7 in the TLP header. See rules for <b>1st DW BE</b> in the PCI Express Specification.
Address	0x00000000: 0xFFFFFFFF		Bytes 8-11 in the TLP header in case of 32-bit TLP.
Requester_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Requester ID in the TLP.
Requester_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Requester ID in the TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of the TLP.

#### Example:

Create a TLP IO event in recording rules, which corresponds to read one DWORD of data from address 0x1000 of the IO address space.

```

Packet = TLP {
    Channels = All
    CurrentState = State_Global
    Action = ( Trigger )
    TLPType = IO
    Address = 0x1000
}

```

#### 4.3.2.4 TLPType = Cfg

Parameter	Value	Default	Comment
BE_First	0:15		Byte 7 in the TLP header. See rules for <b>Last DW BE</b> in the PCI Express Specification.
BE_Last	0:15		Byte 7 in the TLP header. See rules for <b>1st DW BE</b> in the PCI Express Specification.
Requester_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Requester ID in the TLP.
Requester_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Requester ID in the TLP.
Register	0:3F		Bits 2:7 of byte 11 in the TLP header.
Extended_Register	0:F		Bits 0:3 of byte 10 in the TLP header.
Device_Bus	0x00:0xFF		Bytes 8 and 9 in the TLP. Bus Number part of Device ID in the TLP.
Device_Device	0x00:0x1F		Bytes 8 and 9 in the TLP. Device Number part of Device ID in the TLP.
Device_Func	0x0:0x7		Bytes 8 and 9 in the TLP. Function Number part of Device ID in the TLP.
Type_0_1	Type_0 Type_1		Bit 0 of Byte 0 in the TLP header. Type of Cfg TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of the TLP.

#### Example:

This example creates TLP Cfg event in recording rules, that corresponds to read the Capability Pointer from the device's configuration space (**Device ID: Bus Number 0, Device Number 2, Function Number 4**).

```

Packet = TLP {
    TLPType = Cfg
    Device_Bus = 0
    Device_Device = 2
    Device_Func = 4
    Register = 0x34
    Length = 1
    BE_First = 0x1
}

```

#### **4.3.2.5 TLPType = TCfg**

This TLP type has only common for all TLPs fields (please see section 2.1.2.1)

### 4.3.2.6 TLPType = Msg

Parameter	Value	Default	Comment
Routing	To_Root_Complex_Routing Address_Routing Device_ID_Routing Broadcast Local_Routing Gathered_Routing		<b>Message Routing</b> affects the <b>Type</b> field of TLP header. (Bits 2:0).
Message_Code	Assert_INTA Assert_INTB Assert_INTC Assert_INTD Deassert_INTA Deassert_INTB Deassert_INTC Deassert_INTD PM_PME PME_Turn_Off PM_Active_State_Nak PME_TO_Ack ERR_COR ERR_NONFATAL ERR_FATAL Unlock Set_Slot_Power_Limit Vendor_Defined_Type0 Vendor_Defined_Type1 PTM_Request PTM_Response Attention_Indicator_On Attention_Indicator_Blink Attention_Indicator_Off Power_Indicator_On Power_Indicator_Blink Power_Indicator_Off Attention_Button_Pressed Latency_Tolerance_Reporting Optimized_Buffer_Flush_Fill  Direct numeric values can also be used.		Byte 7 in the TLP Header
Requester_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Requester ID in the

			TLP.
Requester_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Requester ID in the TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of TLP.

### 4.3.2.7 TLPType = Cpl

Parameter	Value	Default	Comment
Requester_Bus	0x00:0xFF		Bytes 8 and 9 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 8 and 9 in the TLP. Device Number part of Requester ID in the TLP.
Requester_Func	0x0:0x7		Bytes 8 and 9 in the TLP. Function Number part of Requester ID in the TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of the TLP.
Completer_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Completer ID in the TLP.
Completer_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Completer ID in the TLP.
Completer_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Completer ID in the TLP.
Completion_Status	Successful Unsupported_Request Configuration_Request Completer Abort		Indicates the completion status.
Byte_Count	0:4095		Remaining byte count for the request
Byte_Count_Modifier	0:1		<b>Byte Count Modified:</b> 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.
Address_LO	0:63		Lower byte address for the starting byte of the completion

#### Example 1:

This example shows how to create a Completion TLP event in recording rules. This Completion TLP has 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
    Requester_Bus = 0
    Requester_Device = 0
    Requester_Func = 0
    Completer_Bus = 0
    Completer_Device = 1
    Completer_Func = 0
    Tag=4
    Completion_Status = Unsupported_Request
}

```

Example 2:

This example shows how to create a Completion with Data TLP event in recording rules. This Completion TLP has 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 = Cpl
    Requester_Bus = 0
    Requester_Device = 0
    Requester_Func = 0
    Completer_Bus = 0
    Completer_Device = 1
    Completer_Func = 0
    Tag=4
    Completion_Status = Successful
    Byte_Count = 32
    TLP_Data = ( 0x00000001, 0x00000002, 0x00000003, 0x00000004,
                0x00000005, 0x00000006, 0x00000007, 0x00000008 )
}
```

#### 4.3.2.8 TLPType = FetchAdd, Swap, CAS (32 bit and 64 bit)

Parameter	Value	Default	Comment
Requester_Bus	0x00:0xFF		Bytes 4 and 5 in the TLP. Bus Number part of Requester ID in the TLP.
Requester_Device	0x00:0x1F		Bytes 4 and 5 in the TLP. Device Number part of Requester ID in the TLP.
Requester_Func	0x0:0x7		Bytes 4 and 5 in the TLP. Function Number part of Requester ID in the TLP.
Tag	0x00:0xFF		Byte 6 in the TLP. Tag of the TLP.
BE_First	0:15		Byte 7 in the TLP header. See rules for <b>Last DW BE</b> in the PCI Express Specification.
BE_Last	0:15		Byte 7 in the TLP header. See rules for <b>1st DW BE</b> in the PCI Express Specification.
Address (for 32-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 8-11 in the TLP header in case of 32-bit TLP.
Address_HI (for 64-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 8-11 in the TLP header in case of 64-bit TLP.
Address_LO (for 64-bit TLP)	0x00000000: 0xFFFFFFFF		Bytes 12-15 in the TLP header in case of 64-bit TLP.

#### Example:

This example shows how to create a 32-bit Swap TLP event in recording rules with trigger action.

```

Packet = TLP {
    Channels = All
    CurrentState = State_Global
    Action = ( Trigger )
    TLPType = Swap
    Fmt = 3DW_with_data
    BE_Last = 0xF
    BE_First = 0xF
    Address = 0x1000
    TLP_Data = ( 0x2 )
}

```

### 4.3.3 RecEvent = DLLP

This command creates a DLLP event in recording rules.

#### 4.3.3.1 Common for all types of DLLP's fields

Parameter	Values	Default	Comment
DLLPType	Ack_Nak Data_Link_Feature Vendor Init_FC1 Update_FC Init_FC2 MR_Init MR_Init_FC1 MR_Init_FC2 MR_Update_FC MR_Reset NOP PM Ack Nak		First byte in the DLLP. When not specified, it matches to any type of DLLP packet.
DLLP_CRC	0: 65535		Bytes 4-5 in the DLLP. When not specified, it matches to any CRC value of DLLP packet.

#### 4.3.3.2 DLLPType = Ack\_Nak, Ack, Nak

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

##### Example:

This example creates DLLP event in recording rules that corresponds to Ack or Nak DLLP packet, with a sequence number equal to 120.

```
RecEvent = DLLP {  
    DLLPType = Ack_Nak  
    SeqNum = 120  
}
```

### 4.3.3.3 DLLPType = InitFC1, InitFC2, UpdateFC

Parameter	Values	Default	Comment
Credit_Type	CREDIT_TYPE_P CREDIT_TYPE_NP CREDIT_TYPE_CPL		Specifies DLLP credit type (P, NP or Cpl)
VC_ID	0:7		Virtual Channel, bits 2:0 in the first byte of the DLLP
Hdr_FC	0:255		Contains the credit value for headers of the indicated type (P, NP, or Cpl)
Hdr_Scale	0:3		Contains the scaling factor value for headers of the indicated type (P, NP, or Cpl)
Data_FC	0:4095		Contains the credit value for payload Data of the indicated type (P, NP, or Cpl)
Data_Scale	0:3		Contains the scaling factor value for payload Data of the indicated type (P, NP, or Cpl)

#### Example:

The following example creates DLLP Init FC1 event in recording rules, with credits for VC 0. Credit value for headers is 2. Credit value for data payload is 0.

```

RecEvent = DLLP {
    DLLPType = Init_FC1
    VC_ID = 0
    Hdr_FC = 2
    Data_FC = 0
}

```

#### 4.3.3.4 DLLPType = Vendor, NOP

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

Example:

```
RecEvent = DLLP {  
    DLLPType = Vendor  
    Vendor_Data = 0x010203  
}
```

#### 4.3.3.5 DLLPType = Data\_Link\_Feature

Parameter	Values	Default	Comment
Feature_Ack	Yes, No		Bit to indicate that the transmitting Port has received a Data Link Feature, bit 7 of the byte 1 in the DLLP
Feature_Support	Scaled_Flow_Control, Feature_Support None		Indicate the features supported by the transmitting Port, bytes 1-3 in the DLLP

Example:

```
RecEvent = DLLP {  
    DLLPType = Data_Link_Feature  
    Feature_Ack = Yes  
    Feature_Support = Scaled_Flow_Control  
}
```

### 4.3.3.6 DLLPType = MR\_Init

Parameter	Values	Default	Comment
Phase	Yes, No		Bit 7 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
VHFC	Yes, No		Bit 6 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Mixed_Type	Yes, No		Bit 4 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Authorized	Yes, No		Bit 7 of byte 2 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Dev_Port_Type	PCIE_EndP, Legacy_EndP, Root, Upstream_Switch, Downstream_Switch, PCIE_to_PCI_or_PCI_X_Bridge, PCI_or_PCI_X_to_PCIE_Bridge, Root_Complex_Integrated_EndP, Root_Complex_Event_Collector		Bits 0:3 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Protocol_Version			Bits 4:6 of byte 2 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Max_VL			Bits 0:2 of byte 2 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Max_VH			Byte 3 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"

#### Example:

```

RecEvent = DLLP {
    DLLPType = MR_Init
    Phase = Yes
    VHFC = Yes
    Dev_Port_Type = Root
}

```

#### 4.3.3.7 DLLPType = MR\_Init\_FC1, MR\_Init\_FC2, MR\_UpdateFC

Parameter	Values	Default	Comment
VL_Num	0:7		Bits 0:2 of byte 0 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
VH_Omitted	Yes, No		Bit 7 of byte 2 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
MR_TLP_Type	CREDIT_TYPE_P, CREDIT_TYPE_NP, CREDIT_TYPE_CPL		Specifies type (P, NP, CPL). For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
MR_Credit_Type	Data, Header		Specifies if credit type is header or data. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Data_7_0	0:255		Byte 3 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Data_10_8	0:7		Byte 2 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"

#### Example:

```

RecEvent = DLLP {
    DLLPType = MR_Init
    Phase = Yes
    VHFC = Yes
    Dev_Port_Type = Root
}

```

#### 4.3.3.8 DLLPType = MR\_Reset

Parameter	Values	Default	Comment
VH_Group	0:15		Bits 0:3 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Ack_Request	Ack, Request Any		Bit 7 of byte 1 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"
Assert			Bytes 2, 3 of the DLLP. For more detailed info please see "Multi-Root I/O Virtualization and Sharing Specification"

#### Example:

```
RecEvent = DLLP {  
    DLLPType = MR_Reset  
    VH_Group = 1  
    Ack_Request = Ack  
}
```

#### 4.3.3.9 DLLPType = PM

Parameter	Values	Default	Comment
PM_Type	Enter_L1, Enter_L23, Active_State_request_L1, Request_Ack		Type of power management DLLP, bits 3:7 of Byte 0 in the DLLP

Example:

```
RecEvent = DLLP {  
    DLLPType = PM  
    PM_Type = Enter_L1  
}
```

### 4.3.4 RecEvent = OrderedSet

This command creates Ordered Set and Framing Token events in recording rules.

Each parameter of this event is a flag and can be used independently to select Ordered Set or Framing Token.

Parameter	Values	Default	Comment
SKP	Yes, No	No	Flag for Skip Ordered Set event.
CTRL_SKP	Yes, No	No	Flag for Control Skip Ordered Set event.
EIOS	Yes, No	No	Flag for Electrical Idle Ordered Set event.
EDS	Yes, No	No	Flag for End of Data Stream Framing Token event.
EIEOS	Yes, No	No	Flag for Electrical Idle Exit Ordered Set event.
FTS	Yes, No	No	Flag for Fast Training Sequence event.
SDS	Yes, No	No	Flag for Start of Data Stream Ordered Set event.
EDB	Yes, No	No	Flag for EnD Bad Framing Token event.

#### Example:

The following example created FTS ordered set event in recording rules:

```
RecEvent = OrderedSet {
    FTS = Yes
}
```

### 4.3.5 RecEvent = TS1, TS2

**TS1** and **TS2** modifiers are used to create TS1 and TS2 events in recording rules. Different patterns can be specified for 2.5/5.0 GT/s and 8.0/16.0 GT/s speeds.

#### 4.3.5.1 Common TS1 and TS2 parameters

These parameters are applicable for both TS1 and TS2 Ordered Sets.

Parameters **Use\_G1G2** and **Use\_G3G4** can be used to select speeds at which TS Ordered Sets will be matched.

**G1G2\_\*** parameters are related to pattern that will be used at 2.5/5.0 GT/s and **G3G4\_\*** parameters are related to pattern that will be used at 8.0/16.0 GT/s speeds.

Parameter	Values	Default	Comment
Use_G1G2	Yes, No	Yes	If set to <b>Yes</b> this event will match TS ordered sets at 2.5 and 5.0 GT/s according to the specified pattern.
Use_G3G4	Yes, No	Yes	If set to <b>Yes</b> this event will match TS ordered sets at 8.0 and 16.0 GT/s according to the specified pattern.
G1G2_COM	0:0xFF	0xBC	Byte 0. When operating at 2.5 or 5.0 GT/s: COM (K28.5) for Symbol alignment.
G1G2_LinkNumber	0:0xFF		Byte 1. Link Number within Component.
G1G2_LaneNumber	0:0xFF		Byte 2. Lane Number within Port.
G1G2_N_FTS	0:0xFF		Byte 3. The number of fast training ordered sets required by the Receiver to obtain reliable bit and symbol lock.
G1G2_G1Supported	0:1		Bit 1 of byte 4. 2.5 GT/s Data Rate Supported bit.
G1G2_G2Supported	0:1		Bit 2 of byte 4. 5.0 GT/s Data Rate Supported bit.
G1G2_G3Supported	0:1		Bit 3 of byte 4. 8.0 GT/s Data Rate Supported bit.
G1G2_G4Supported	0:1		Bit 4 of byte 4. 16.0 GT/s Data Rate Supported bit.
G1G2_AutonomusChange	0:1		Bit 6 of byte 4. Autonomous Change/Selectable De-emphasis bit.

G1G2_SpeedChange	0:1		Bit 7 of byte 4.
G1G2_HotReset	0:1		Bit 0 of byte 5.
G1G2_DisableLink	0:1		Bit 1 of byte 5.
G1G2_Loopback	0:1		Bit 2 of byte 5.
G3G4_DisableScrambling	0:1		Bit 3 of byte 5.
G3G4_LinkNumber	0:0xFF		See Comments for <b>G1G2_*</b> parameters.
G3G4_LaneNumber	0:0xFF		
G3G4_N_FTS	0:0xFF		
G3G4_G1Supported	0:1		
G3G4_G2Supported	0:1		
G3G4_G3Supported	0:1		
G3G4_G4Supported	0:1		
G3G4_AutonomusChange	0:1		
G3G4_SpeedChange	0:1		
G3G4_HotReset	0:1		
G3G4_DisableLink	0:1		
G3G4_Loopback	0:1		

### 4.3.5.2 TS1 Parameters

Parameter	Values	Default	Comment
G1G2_ComplianceReceive	0:1		Bit 4 of byte 5.
G3G4_TS1SymbolAlignment	0:0xFF	0x1E	Byte 0. When operating at 8.0 GT/s or above: Encoded as 1Eh (TS1 Ordered Set).
G3G4_ComplianceReceive	0:1		Bit 4 of byte 5.
G3G4_EqualizationControl	0:3		Bits 1:0 of byte 6.
G3G4_ResetEIEOSIntervalCount	0:1		Bit 2 of byte 6.
G3G4_TransmitterPreset	0:0xF		Bits 6:3 of byte 6.
G3G4_UsePreset	0:1		Bit 7 of byte 6.
G3G4_PreCursorCoefficient	0:0x3F		Bits 5:0 of byte 7. FS when the EC field of Symbol 6 is 01b. Otherwise, Pre-cursor Coefficient for the current data rate of operation.
G3G4_RetimerEqualizationExtend	0:1		Bit 7 of byte 7.
G3G4_CursorCoefficient	0:0x3F		Bits 5:0 of byte 8. LF when the EC field of Symbol 6 is 01b. Otherwise, Cursor Coefficient for the current data rate of operation.
G3G4_PostCursorCoefficient	0:0x3F		Bit 5:0 of byte 9. Post-cursor Coefficient for the current data rate of operation.
G3G4_RejectCoefficientValue	0:1		Bit 6 of byte 9.
G3G4_Parity	0:1		Bit 7 of byte 9. Parity (P). This bit is the even parity of all bits of Symbols 6, 7, and 8 and bits 6:0 of Symbol 9.

**Note:**

When operating at 2.5 or 5.0 GT/s: Symbols 6-13 will be TS1 Identifier (0x4A) and mask for Symbols 14-15 will be set to 0.

When operating at 8.0 GT/s or above: Symbols 10-13 will be TS1 Identifier (0x4A) and mask for Symbols 14-15 will be set to 0.

### 4.3.5.3 TS2 Parameters

Parameter	Values	Default	Comment
G1G2_RetimerPresent	0:1		Bit 4 of byte 5. 0b No Retimers present 1b One or more Retimers present
G3G4_TS2SymbolAlign	0:0xFF	0x2D	Byte 0. When operating at 8.0 GT/s or above: Encoded as 2Dh (TS2 Ordered Set).
G3G4_RetimerPresent	0:1		Bit 4 of byte 5. 0b No Retimers present 1b One or more Retimers present
G3G4_TwoRetimersPresent	0:1		Bit 5 of byte 5. 0b Zero or one Retimers present 1b Two or more Retimers present
G3G4_EqualizationRequestDataRate	0:3		Bits 5:4 of byte 6. 00b indicates 8.0 GT/s Data Rate 10b indicates 16.0 GT/s Data Rate. 01b indicates 32.0 GT/s Data Rate. 11b Reserved.
G3G4_QuiesceGuarantee	0:1		Bit 6 of byte 6.
G3G4_RequestEqualization	0:1		Bit 7 of byte 6.

**Note:**

When operating at 2.5 or 5.0 GT/s: Symbols 6-13 will be TS2 Identifier (0x45) and mask for Symbols 14-15 will be set to 0.

When operating at 8.0 GT/s or above: Symbols 7-13 will be TS2 Identifier (0x45) and mask for Symbols 14-15 will be set to 0.

### 4.3.6 RecEvent = Skip

This command creates Skip ordered set event in recording rules.

Parameter	Values	Default	Comment
IsAnySkip	Yes, No	This parameter must be defined	If this parameter set to <b>Yes</b> then this event will match the Skip of any size at any speed. In this case other parameters cannot be specified. If this parameter set to <b>No</b> then this event will match Skip of standard 4 dword size only at 8.0 and 16.0 GT/s speeds. In this case <b>SkpEnd</b> and <b>LSFR</b> may be specified.
SkpEnd	0:0xFF	0xE1	Byte 12. SKP_END Symbol. Signifies the end of the SKP Ordered Set after three more Symbols.
LSFR	0:0xFFFFFFFF		Bytes 13-15. This field also includes Data Parity (bit 0 in byte 13).

**Note:** If **IsAnySkip** parameter is set to **No** only the fourth dword of pattern can be changed. First three dwords will be filled with SKP Symbols (0xAA).

Example:

```
RecEvent = Skip {
    IsAnySkip = No
    SkpEnd = 0xE1
}
```

### 4.3.7 RecEvent = CtrlSkip

This command creates Control Skip ordered set event in recording rules.

Parameter	Values	Default	Comment
IsAnyCtrlSkip	Yes No	This parameter must be defined	If this parameter set to <b>Yes</b> then this event will match the Control Skip of any size at 16.0 GT/s speed. If this parameter set to <b>No</b> then this event will match Control Skip of standard 4 dword size only at 16.0 GT/s speed. In this case <b>SkipEndCtrl</b> and <b>Payload, MarginParity</b> and other parameters may be specified.
SkipEndCtrl	0:0xFF	0x78	Byte 12. SKP_END_CTL Symbol. Signifies the end of the Control SKP Ordered Set after three more Symbols.
Payload	0:0xFF		Byte 15.
DataParity	0:1		Bit 7 of byte 13.
FirstRetimerParity	0:1		Bit 6 of byte 13.
SecondRetimerParity	0:1		Bit 5 of byte 13.
CRC	0:0x1F		Bits 4:0 of byte 13.
MarginParity	0:1		Bit 7 of byte 14.
UsageModel	0:1		Bit 6 of byte 14.
MarginType	0:7		Bits 5:3 of byte 14.
ReceiveNumber	0:7		Bits 2:0 of byte 14.

**Note:** If **IsAnyCtrlSkip** parameter is set to **No** only the fourth dword of pattern can be changed. First three dwords will be filled with SKP Symbols (0xAA).

#### Example:

```

RecEvent = CtrlSkip {
    IsAnySkip = No
    SkpEndCtrl = 0x78
    Payload = 0x01
}

```

### 4.3.8 RecEvent = Lane\_Margining

This command creates Lane Margining event in recording rules.

The Margin Type field together with a Receiver Number and specific Margin Payload field define various commands used for margining. Parameter **Margin\_Command** can be used to select Margin Command. It has predefined list of supported commands.

If **Margin\_Command** is defined it must be before **MarginType**, **Payload** and **Receiver\_Number** parameters. Specified value of each of these parameters will be checked and it must be supported by the selected Margin Command.

If **Margin\_Command** is omitted then **MarginType**, **Payload** and **Receiver\_Number** fields can have any values.

For a list of supported values by Margin Commands consult "Margin Commands and Corresponding Responses" table in the PCI Express Base Specification.

Parameter	Values	Default	Comment
Margin_Command	No_Command, Access_Retimer_register, Report_Margin_Control_Capabilities, Report_M_NumVoltageSteps, Report_M_NumTimingSteps, Report_M_MaxTimingOffset, Report_M_MaxVoltageOffset, Report_M_SamplingRateVoltage, Report_M_SamplingRateTiming, Report_M_SampleCount, Report_M_MaxLanes, Report_Reserved, Set_Error_Count_Limit, Go_to_Normal_Settings, Clear_Error_Log, Step_Margin_timing_to_offset, Step_Margin_voltage_to_offset, Vendor_Defined		
Payload	0:0xFF	See note	Byte 15.
MarginType	0:7	See note	Bits 5:3 of byte 14.
Receiver_Number	Broadcast (000b), Rx_A (001b), Rx_B (010b), Rx_C (011b), Rx_D (100b), Rx_E (101b), Rx_F (110b), Reserved (111b) or 0:7	See note	Bits 2:0 of byte 14.
SkipEndCtrl	0:0xFF	0x78	Byte 12. SKP_END_CTL Symbol. Signifies the end of the Control SKP Ordered Set after three more

			Symbols.
DataParity	0:1		Bit 7 of byte 13.
FirstRetimerParity	0:1		Bit 6 of byte 13.
SecondRetimerParity	0:1		Bit 5 of byte 13.
CRC	0:0x1F		Bits 4:0 of byte 13.
MarginParity	0:1		Bit 7 of byte 14.
UsageModel	0:1		Bit 6 of byte 14. 0b: Lane Margining at Receiver 1b: Reserved Encoding

**Note:**

Default value of **MarginType**, **Payload** and **Receiver\_Number** parameters will be calculated as follow:

- If **Margin\_Command** is omitted, then mask will be set to 0;
- If several values are supported by the parameter for the selected Margin Command, then the smallest possible value will be used;
- If only one value is supported for the selected Margin Command, then it will be used.

Only the fourth dword of pattern can be changed. First three dwords will be filled with SKP Symbols (0xAA).

Example:

The following example creates Lane Margining event in Recording Options for Margin Command Set\_Error\_Count\_Limit. In this example **Payload** will be 0xC0 and **MarginType** will be 0b010.

```

RecEvent = Lane_Margining {
    Margin_Command = Set_Error_Count_Limit
    Receiver_Number = Rx_A
    MarginParity = 1
}

```

### 4.3.9 RecEvent = LinkState

This command creates Link State event in recording rules.

Each parameter of this event is a flag and can be used independently to select Link State event.

**LinkWidthSwitch\_From\_X\*** and **LinkWidthSwitch\_To\_X\*** can be enabled only if device supports required link width. For example, **LinkWidthSwitch\_From\_X16** and **LinkWidthSwitch\_To\_X16** events cannot be selected for T48 in Standard Mode or for T416 in Multiport Mode.

Parameter	Values	Default	Comment
EnterEI	Yes, No	No	Enter Electrical Idle event.
ExitEI	Yes, No	No	Exit Electrical Idle event.
CLKREQ_Asserted	Yes, No	No	CLKREQ# Asserted event.
CLKREQ_Deasserted	Yes, No	No	CLKREQ# Deasserted event.
WAKE_Asserted	Yes, No	No	WAKE# Asserted event.
WAKE_Deasserted	Yes, No	No	WAKE # Deasserted event.
PERST_Asserted	Yes, No	No	PERST# Asserted event.
PERST_Deasserted	Yes, No	No	PERST # Deasserted event.
SpeedSwitch_From_G1	Yes, No	No	Speed Switch from 2.5 GT/s to any speed.
SpeedSwitch_From_G2	Yes, No	No	Speed Switch from 5.0 GT/s to any speed.
SpeedSwitch_From_G3	Yes, No	No	Speed Switch from 8.0 GT/s to any speed.
SpeedSwitch_From_G4	Yes, No	No	Speed Switch from 16.0 GT/s to any speed.
SpeedSwitch_To_G1	Yes, No	No	Speed Switch to 2.5 GT/s from any speed.
SpeedSwitch_To_G2	Yes, No	No	Speed Switch to 5.0 GT/s from any speed.
SpeedSwitch_To_G3	Yes, No	No	Speed Switch to 8.0 GT/s from any speed.
SpeedSwitch_To_G4	Yes, No	No	Speed Switch to 16.0 GT/s from any speed.
LinkWidthSwitch_From_X1	Yes, No	No	Link Width Switch from X1 to any Link Width.
LinkWidthSwitch_From_X2	Yes, No	No	Link Width Switch from X2 to any Link Width.
LinkWidthSwitch_From_X4	Yes, No	No	Link Width Switch from X4 to any Link Width.
LinkWidthSwitch_From_X8	Yes, No	No	Link Width Switch from X8 to any Link Width.
LinkWidthSwitch_From_X16	Yes, No	No	Link Width Switch from X16 to any Link Width.
LinkWidthSwitch To X1	Yes,	No	Link Width Switch to X1 from any Link

	No		Width.
LinkWidthSwitch_To_X2	Yes, No	No	Link Width Switch to X2 from any Link Width.
LinkWidthSwitch_To_X4	Yes, No	No	Link Width Switch to X4 from any Link Width.
LinkWidthSwitch_To_X8	Yes, No	No	Link Width Switch to X8 from any Link Width.
LinkWidthSwitch_To_X16	Yes, No	No	Link Width Switch to X16 from any Link Width.

Example:

The following example creates Link State event in Recording Options for Speed Switch from 2.5 or 5.0 GT/s to any speed *or* Speed Switch from any speed to 8.0 GT/s:

```
RecEvent = LinkState {  
    SpeedSwitch_From_G1 = Yes  
    SpeedSwitch_From_G2 = Yes  
    SpeedSwitch_To_G3 = Yes  
}
```

### 4.3.10 RecEvent = Error

This command creates Error event in recording rules.

Each parameter of this event is a flag and can be used independently to select error types. At least one error type must be selected.

Parameter	Values	Default	Comment
DisparityError	Yes, No	No	Flag for Disparity Error type.
SymbolError	Yes, No	No	Flag for Symbol Error type.
BlockAlignError	Yes, No	No	Flag for Block Alignment Error type.
IdleSymbolError	Yes, No	No	Flag for Idle Symbol Error type.

#### Example:

The following example creates Error event for Symbol Error:

```
RecEvent = Error {  
    SymbolError = Yes  
}
```

### 4.3.11 RecEvent = NVMe\_Register

This command creates NVMe Register event in recording rules.

To create NVMe Register event either **Type** or **Offset** parameter must be specified.

Parameter	Values	Default	Comment
BAR_0	0:0xFFFFFFFF	0	Lower 32 bits of MBAR.
BAR_1	0:0xFFFFFFFF	0	Higher 32 bits of MBAR.
Type	Custom ACQ AQA ASQ CAP CC CQyHDBL CSTS INTMC INTMS NSSR BPINFO BPRSEL BPMBL Reserved_0x18_0x1B Reserved_0x50_0xEFF Reserved_Command_Set_Specific SQyTDBL VS CMBLOC CMBSZ	Custom	Type of NVMe Register. This parameter sets offset according to the selected type.
Offset	0:0xFFFFFFFF	Must be specified if <b>Type</b> is <b>Custom</b>	Offset of NVMe Register. This field can be specified only if <b>Type</b> is <b>Custom</b> .
Queue	0:0xFFFF	0	Number of Queue. Only for doorbell registers.
DSTRD	$2^N$ , $2 \leq N \leq 17$	4	Doorbell Stride. Only for doorbell registers.
Data_Pattern	0:0xFFFFFFFF		The first dword of data. Events with <b>Data_Pattern</b> defined will match only register writes.

#### Example:

The following example creates NVMe Register event:

```
RecEvent = NVMe_Register {
    BAR_0 = 0xff00
    Type = SQyTDBL
    Queue = 10
    DSTRD = 16
    Data_Pattern = "0x10"
```

```
}
```

### 4.3.12 RecEvent = NVMe\_Command\_Submission

This command creates NVMe Command Submission event in recording rules.

Reserved fields:

- bits 13:10 of dword 0,
- dword 2,
- dword 3

will be filled with 0.

Parameter	Values	Default	Comment
Command_Identifier	0:0xFFFF		Bits 31:16 of the first dword. This field specifies a unique identifier for the command when combined with the Submission Queue identifier.
PRP_or_SGL_data_transfer	PRP_data_transfer (0b00) SGL_data_transfer physically_contiguous_buffer (0b01) SGL_data_transfer_SGL_descriptor (0b10) Reserved_PSDT (0b11) or 0:3	PRP_data_transfer for admin <b>Opcode</b> values	Bits 15:14 of the first dword. This field specifies whether PRPs or SGLs are used for any data transfer associated with the command.
FUSE	Normal_Operation (0b00) Fused_Operation_First_Command (0b01) Fused_Operation_Second_Command (0b10) Reserved_FUSE (0b11) or 0:3	Normal_Operation	Bits 9:8 of the first dword. This parameter specifies whether this command is part of a fused operation and if so, which command it is in the sequence.
Opcode	Delete_IO_Submission_Queue Create_IO_Submission_Queue Get_Log_Page Delete_IO_Completiom_Queue Create_IO_Completiom_Queue Identify Abort Set_Features Get_Features Asynchronous_Event_Request Firmware_Activate Firmware_Image_Download NVMe_MI_Send NVMe_MI_Receive Device_Self_test Directive_Receive Directive_Send Doorbell_Buffer_Config		Bits 7:0 of the first dword. This parameter specifies the opcode of the command to be executed.

	Format_NVM Security_Send Security_Receive Virtualization_Management Sanitize IO_Command_Specific Admin_Command_Vendor_Specific Flush Write Read Write_Uncorrectable Compare Dataset_Management NVMe_Command_Vendor_Specific <i>or</i> 0:0xFF		
Namespace_Id	0:0xFFFFFFFF		Dword 1. This field specifies the namespace that this command applies to.
Metadata_Pointer_LO	0:0xFFFFFFFF		Dword 4. Lower dword of Metadata Pointer.
Metadata_Pointer_HI	0:0xFFFFFFFF		Dword 5. Higher dword of Metadata Pointer.
PRP_Entry_1_LO	0:0xFFFFFFFF		Dword 6. Lower dword of PRP Entry 1.
PRP_Entry_1_HI	0:0xFFFFFFFF		Dword 7. Higher dword of PRP Entry 1.
PRP_Entry_2_LO	0:0xFFFFFFFF		Dword 8. Lower dword of PRP Entry 2.
PRP_Entry_2_HI	0:0xFFFFFFFF		Dword 9. Higher dword of PRP Entry 2.
Command_Dwords	Array (maximum 6 dwords) Each element is 0:0xFFFFFFFF		Dwords 10-15. Command specific.

Example:

The following example creates NVMe Command Submission event:

```

RecEvent = NVMe_Command_Submission {
  Opcode = NVMe_MI_Send
  Metadata_Pointer_LO = "0x00000000"
  Command_Dwords = (
    "0xffXXXXXXXX", "0xFFFFFFFF00", "0xFFFFFFFFXX", "0xFFFFFFFFXX",
    "0xFFFFFFFFXX", "0xFFFFFFFFXX"
  )
}

```

### 4.3.13 RecEvent = NVMe\_Command\_Completion

This command creates NVMe Command Completion event in recording rules.

Reserved fields:

- dword 1,
- bits 29:28 of dword 3

will be filled with 0.

Parameter	Values	Default	Comment
Command_Specific	0:0xFFFFFFFF		The first dword. The contents of Dword 0 are command specific.
SQ_Identifier	0:0xFFFF		Bits 31:16 of dword 2. This field indicates the Submission Queue to which the associated command was issued.
SQ_Head_Pointer	0:0xFFFF		Bits 15:0 of dword 2. This field indicates the current Submission Queue Head pointer for the Submission Queue indicated in the SQ Identifier field.
Do_Not_Retry	If Re_issued_Expected_To_Succeed (0) If Re_issued_Expected_To_Fail (1) or 0:1		Bit 31 of dword 3.
More	No_Additional_Status_Information_Available (0) Additional_Status_Information_Available (1) or 0:1		Bit 30 of dword 3.
Status_Code_Type	Generic_Command_Status (0) Command_Specific_Status (1) Media_Error (2) Vendor_Specific (7) or 0:7		Bits 27:25 of dword 3. This field indicates the status code type.
Status_Code	Successful_Completion Invalid_Command_Opcode Invalid_Field_in_Command Command_ID_Conflict Data_Transfer_Error Commands_Aborted_due_to_Power_Loss_Notification Internal_Device_Error Command_Abort_Requested Command_Aborted_due_to_SQ_Deletion Command Aborted due to Failed Fused Command		Bits 24:17 of dword 3. This field indicates a status code identifying any error or status information for the command indicated.

	Command_Aborted_due_to_Missing_Fused_Command Invalid_Namespace_or_Format Command_Sequence_Error Invalid_SGL_Segment_Descriptor Invalid_Number_of_SGL_Descriptors Data_SGL_Length_Invalid Metadata_SGL_Length_Invalid SGL_Descriptor_Type_Invalid Invalid_Use_of_Controller_Memory_Buffer PRP_Offset_Invalid Atomic_Write_Unit_Exceeded Operation_Denied SGL_Offset_Invalid Host_Identifier_Inconsistent_Format Keep_Alive_Timeout_Expired Keep_Alive_Timeout_Invalid Command_Aborted_due_to_Preempt_and_Abort Sanitize_Failed Sanitize_In_Progress SGL_Data_Block_Granularity_Invalid Command_Not_Supported_for_Queue_in_CMB LBA_Out_of_Range Capacity_Exceeded Namespace_Not_Ready IO_Command_Set_Specific Vendor_Specific or 0:0xFF		
Phase_Tag	0:1		Bit 16 of dword 3. This field identifies whether a Completion Queue entry is new.
Command_Identifier	0:0xFFFF		Bits 15:0 of dword 3. This field indicates the identifier of the command that is being completed.

Example:

The following example creates NVMe Command Completion event:

```

RecEvent = NVMe_Command_Completion {
    Command_Specific = "0xFFFFaaaa"
    Do_Not_Retry = If_Re_issued_Expected_To_Succeed
    More = No_Additional_Status_Information_Available
    Status_Code_Type = Generic_Command_Status
    Status_Code = Successful_Completion
    Phase_Tag = "0x0"
}

```

### 4.3.14 RecEvent = MCTP

This command creates MCTP event in recording rules.

Common parameters for all MCTP packets:

Parameter	Values	Default	Comment
MCTP_Bus	VDM_Only SMBus_Only VDM_And_SMBus	VDM_And_SMBus	Enables VDM and/or SMBus transport(s). SMBus transport may be enabled only if "SMBus Recording" is checked on "General" page of Recording Options.
VDM_IC	0:1		Bit 7 of the first byte of the MCTP packet payload. Indicates whether the MCTP message is covered by an overall MCTP message payload integrity check.
SMBus_IC			
VDM_MCTPType	MCTP_Control (0) MCTP_PLDM (1) MCTP_NC_SI (2) MCTP_NVME_MI (4) or 0:0x7F		Bits 6:0 of the first byte of the MCTP packet payload. Defines the type of payload contained in the message data portion of the MCTP message.
SMBus_MCTPType			
VDM_Message_Tag	0:7		Bits 2:0 of byte 3 of the MCTP transport header. Field that, along with the Source Endpoint IDs and the Tag Owner (TO) field, identifies a unique message at the MCTP transport level.
SMBus_Message_Tag			
VDM_Packet_Sequence_Number	0:3		Bits 5:4 of byte 3 of the MCTP transport header. For messages that span multiple packets, the packet sequence number increments modulo 4 on each successive packet.
SMBus_Packet_Sequence_Number			
VDM_Packet_Type	Start_Packet (0b1X) Middle_Packet (0b00) End_Packet (0bX1) Single_Packet (0b11) Or 0:3		Bits 7:6 of byte 3 of the MCTP transport header. It consists of SOM and EOM. SOM (bit 7) is set to 1b if this packet is the first packet of a message. EOM (bit 6) is set to 1b if this packet is the last packet of a message.
SMBus_Packet_Type			
VDM_Source_Eid	0:0xFF		Byte 2 of the MCTP transport header. The EID of the originator of the MCTP packet.
SMBus_Source_Eid			
VDM_Destination_Eid	0:0xFF		Byte 1 of the MCTP transport header. The EID for the endpoint to receive the MCTP packet.
SMBus_Destination_Eid			
VDM_Header_Revision	0:0xF		Bits 3:0 of byte 1 of the MCTP transport header. Identifies the format, physical framing,

SMBus_Header_Revision			and data integrity mechanism used to transfer the MCTP common fields in messages on a given physical medium.
VDM_Message_Data	0:0FFFFFFF		Bytes 1-3 of the MCTP packet payload. Can be specified only if <b>MCTPType</b> parameter is not defined.
SMBus_Message_Data			

Example:

The following example creates MCTP event for VDM and SMBus transports:

```

RecEvent = MCTP {
    MCTP_Bus = VDM_And_SMBus
    VDM_Packet_Type = Start_Packet
    VDM_Source_EId = "0x09"
    VDM_Destination_EId = "0x08"
    SMBus_Packet_Type = Start_Packet
    SMBus_Source_EId = "0x09"
    SMBus_Destination_EId = "0x08"
}

```

#### 4.3.14.1 VDM Parameters

The following parameters are applicable only for VDM transport:

Parameter	Values	Default	Comment
VDM_Length	0:0x3FF		Bits 1:0 of byte 2 and bits 7:0 of byte 3 of the TLP Header. Length of the PCIe VDM Data in dwords.
VDM_Attributes	0:3		Bits 5:4 of byte 2 of the TLP Header.
VDM_Error_Present	0:1		Bit 6 of byte 2 of the TLP Header.
VDM_TLP_Didgest	0:1		Bit 7 of byte 2 of the TLP Header.
VDM_Tag_Bit8	0:1		Bit 3 of byte 1 of the TLP Header.
VDM_Traffic_Class	0:7		Bits 6:4 of byte 1 of the TLP Header.
VDM_Tag_Bit9	0:1		Bit 7 of byte 1 of the TLP Header.
VDM_TLP_Type	0:0x1F		Bits 4:0 of byte 0 of the TLP Header.
VDM_TLP_Format	0:3	0b11	Bits 6:5 of byte 0 of the TLP Header. Set to 11b to indicate 4 dword header with data.
VDM_Message_Code	0:0xFF	0x7F	Byte 7 of the TLP Header. Set to 0111_1111b to indicate a Type 1 VDM.
VDM_MCTP_Code	0:0xF	0	Bits 3:0 of byte 6 of the TLP Header. Value that uniquely differentiates MCTP messages from other DMTF VDMs.
VDM_Pad_Length	0:3		Bits 5:4 of byte 7 of the TLP Header. Pad Length. 1-based count (0 to 3) of the number of 0x00 pad bytes that have been added to the end of the packet to make the packet dword aligned with respect to PCIe.
VDM_PCI_Requester_Id	0:0xFFFF		Bytes 4-5 of the TLP Header. Bus/device/function number of the managed endpoint sending the message.
VDM_Vendor_Id	0:0xFFFF	0x1AB4	Bytes 10-11 of the TLP Header. Set to 6836 (0x1AB4) for DMTF VDMs.
VDM_Target_Id	0:0xFFFF		Bytes 8-9 of the TLP Header. For Route By ID messages, this is the bus/device/function number that is the physical address of the target endpoint.

#### 4.3.14.2 SMBus Parameters

The following parameters are applicable only for SMBus transport:

Parameter	Values	Default	Comment
SMBus_Destination_Slave_Address	0:0x7F		Bits 7:1 of byte 0. SMBus Destination Slave Address.
SMBus_Command_Code	0:0xFF	0x0F	Byte 1. SMBus Command Code. According to the specification all MCTP over SMBus messages use a command code of 0x0F.
SMBus_Byte_Count	0:0xFF		Byte 2. Byte count for the SMBus Block Write protocol transaction that is carrying the MCTP packet content.
SMBus_Source_Slave_Address	0:0x7F		Bits 7:1 of byte 3. SMBus Source Slave Address.
SMBus_Message_Header_And_Data	0:0xFFFFFFFF		Bytes 12-15 of SMBus packet.

### 4.3.14.3 MCTPType = Control

Parameter	Values	Default	Comment
VDM_Control_Class	Control_Request Control_Response		Combination of Tag Owner, Request and Datagram bits.
SMBus_Control_Class	Control_Broadcast_Request Control_Datagram Control_Broadcast_Datagram		
VDM_Instance_ID	0:0x1F		Bits 4:0 of byte 1 of the MCTP packet payload. The Instance ID field is used to identify new instances of an MCTP control Request or Datagram.
SMBus_Instance_ID			
VDM_Control_Command	Set_Endpoint_ID (0x1) Get_Endpoint_ID (0x2) Get_Endpoint_UUID (0x2) Get_MCTP_Version_Support (0x4) Get_Message_Type_Support (0x5) Get_Vendor_Defined_Message_Support (0x6) Resolve_Endpoint_ID (0x7) Allocate_Endpoint_IDs (0x8)		Byte 2 of the MCTP packet payload. For Request messages, this field is a command code indicating the type of MCTP operation the packet is requesting.
SMBus_Control_Command	Routing_Information_Update (0x9) Get_Routing_Table_Entries (0xA) Prepare_For_Endpoint_Discovery (0xB) Endpoint_Discovery (0xC) Discovery_Notify (0xD) Get_Network_ID (0xE) Query_Hop (0xF) Resolve_UUID (0x10) <i>or</i> 0:0xFF		
VDM_Control_Completion_Code	Control_Success (0x0) Control_Error (0x1) Control_Error_Invalid_Data (0x2) Control_Error_Invalid_Length (0x3) Control_Error_Not_Ready (0x4) Control_Error_Unsupported_Cmd (0x5)		Byte 3 of the MCTP packet payload. Only when <b>VDM_Control_Class = Control_Response</b> . This field contains a value that indicates whether the response completed normally or provides information regarding the error condition.
SMBus_Control_Completion_Code	Control_Command_Specific (0x6) Control_Reserved (0x7) <i>or</i> 0:0xFF		

#### Example:

The following example creates MCTP event with **MCTP\_Control** type:

```
RecEvent = MCTP {  
    MCTP_Bus = VDM_Only  
    VDM_MCTPType = MCTP_Control  
    VDM_Control_Class = Control_Request  
    VDM_Control_Command = Get_Endpoint_UUID  
    VDM_Instance_ID = "0x2"  
}
```

#### 4.3.14.4 MCTPType = PLDM

Parameter	Values	Default	Comment
VDM_PLDM_Class	PLDM_Response PLDM_Request PLDM_Reserved PLDM_Unacknowledged		Datagram and Request bit combinations: 00b – For PLDM response messages 01b – For PLDM request messages 10b – Reserved 11b – For Unacknowledged PLDM request messages or asynchronous notifications
SMBUS_PLDM_Class			
VDM_PLDM_Type	Messaging_Control_And_Discovery (0x0) SMBIOS (0x1) Platform_Monitoring_And_Control (0x2) BIOS_Control_And_Configuration (0x3) FRU_Data (0x4)		Bits 5:0 of byte 2 of the MCTP packet payload. The PLDM Type field identifies the type of PLDM that is being used in the control or data transfer carried out using this PLDM message.
SMBus_PLDM_Type	Firmware_Update (0x5) Redfish_enablement (0x6) OEM_Specific (0x3F) or 0:0x3F		
VDM_MC_Command	MC_SetTID (0x1) MC_GetTID (0x2) GetPLDMVersion (0x3) GetPLDMTypes (0x4) GetPLDMCommands (0x5)		When <b>PLDM_Type = Messaging_Control_And_Discovery</b>
SMBus_MC_Command	or 0:0xFF		
VDM_SMBIOS_Command	GetSMBIOSStructureTableMetadata (0x1) SetSMBIOSStructureTableMetadata (0x2) GetSMBIOSStructureTable (0x3) SetSMBIOSStructureTable (0x4) GetSMBIOSStructureByType (0x5) GetSMBIOSStructureByHandle (0x6)		When <b>PLDM_Type = SMBIOS</b>
SMBus_SMBIOS_Command	or 0:0xFF		

VDM_PM_Command	PM_SetTID (0x01) PM_GetTID (0x02) GetTerminusUID (0x03) SetEventReceiver (0x04) GetEventReceiver (0x05) PlatformEventMessage (0x0A) SetNumericSensorEnable (0x10) GetSensorReading (0x11) GetSensorThresholds (0x12) SetSensorThresholds (0x13) RestoreSensorThresholds (0x14) GetSensorHysteresis (0x15) SetSensorHysteresis (0x16) InitNumericSensor (0x17) SetStateSensorEnables (0x20) GetStateSensorReadings (0x21) InitStateSensor (0x22) SetNumericEffectorEnable (0x30) SetNumericEffectorValue (0x31)		When <b>PLDM_Type = Platform_Monitoring_And_Control</b>
SMBus_PM_Command	GetNumericEffectorValue (0x32) SetStateEffectorEnables (0x38) SetStateEffectorStates (0x39) GetStateEffectorStates (0x3A) GetPLDMEventLogInfo (0x40) EnablePLDMEventLogging (0x41) ClearPLDMEventLog (0x42) GetPLDMEventLogTimestamp (0x43) SetPLDMEventLogTimestamp (0x44) ReadPLDMEventLog (0x45) GetPLDMEventLogPolicyInfo (0x46) SetPLDMEventLogPolicy (0x47) FindPLDMEventLogEntry (0x48) GetPDRRepositoryInfo (0x50) GetPDR (0x51) FindPDR (0x52) RunInitAgent (0x58) <i>or</i> 0:0xFF		
VDM_BIOS_Command	GetBIOSTable (0x1) SetBIOSTable (0x2) UpdateBIOSTable (0x3) GetBIOSTableTags (0x4) SetBIOSTableTags (0x5) AcceptBIOSAttributesPendingValues (0x6) SetBIOSAttributeCurrentValue (0x7) GetBIOSAttributeCurrentValueByHandle (0x8)		When <b>PLDM_Type = BIOS_Control_And_Configuration</b>

SMBus_BIOS_Command	GetBIOSAttributePendingValueByHandle (0x9) GetBIOSAttributeCurrentValueByType (0xA) GetBIOSAttributePendingValueByType (0xB) GetDateTime (0xC) SetDateTime (0xE) GetBIOSStringTableStringType (0xE) SetBIOSStringTableStringType (0xF) or 0:0xFF		
VDM_FRU_Command	GetFRURecordTableMetadata (0x1) GetFRURecordTable (0x2) SetFRURecordTable (0x3)		When <b>PLDM_Type = FRU_Data</b>
SMBus_FRU_Command	GetFRURecordByOption (0x4) or 0:0xFF		
VDM_FW_Command	QueryDeviceIdentifiers (0x01) GetDeviceMetaData (0x02) RequestUpdate (0x10) SendPackageData (0x11) RetrieveDeviceData (0x12) PassComponentTable (0x13) UpdateComponent (0x14) RequestFirmwareData (0x15) TransferComplete (0x16)		When <b>PLDM_Type = Firmware_Update</b>
SMBus_FW_Command	VerifyComplete (0x17) ApplyComplete (0x18) RestoreDeviceData (0x19) ActivateFirmware (0x1A) GetStatus (0x1B) CancelUpdateComponent (0x1C) CancelUpdate (0x1D) or 0:0xFF		
VDM_Redfish_Command	NegotiateRedfishParameters (0x01) GetSchemaDictionary (0x02) GetSchemaURI (0x03) GetResourceETag (0x04) SetEventReceiver (0x05) PlatformEventMessage (0x0A) RDEOperationInit (0x10) RedfishCreate (0x11) RedfishRead (0x12) RedfishWrite (0x13) RedfishAction (0x14) RedfishDelete (0x15) RedfishHead (0x16)		When <b>PLDM_Type = Redfish_enablement</b>

SMBus_Redfish_Command	SupplyCustomRequestParameters (0x17) RetrieveCustomResponseParameters (0x18) RDEOperationComplete (0x19) RDEOperationStatus (0x1A) RDEOperationKill (0x1B) RDEOperationEnumerate (0x1C) MultipartSend (0x30) MultipartReceive (0x31) RedfishGetPDRRepositoryInfo (0x50) RedfishGetPDR (0x51) or 0:0xFF		
VDM_PLDM_InstanceId	0:0x1F		Bits 4:0 of byte 1 of the MCTP packet payload. The Instance Identifier field is used to identify a new instance of a PLDM request to differentiate new PLDM requests that are sent to the same PLDM terminus.
SMBus_PLDM_InstanceId			
VDM_PLDM_Header_Version	0:3		Bits 7:6 of byte 2 of the MCTP packet payload. The Header Version field identifies the header format.
SMBus_PLDM_Header_Version			

Example:

The following example creates MCTP event with **MCTP\_PLDM** type:

```

RecEvent = MCTP {
    MCTP_Bus = SMBus_Only
    SMBus_MCTPType = MCTP_PLDM
    SMBUS_PLDM_Class = PLDM_Response
    SMBus_PLDM_Type = Messaging_Control_And_Discovery
    SMBus_PLDM_InstanceId = "0xa"
    SMBus_MC_Command = MC_GetTID
}

```

#### 4.3.14.5 MCTPType = NC\_SI

Parameter	Values	Default	Comment
VDM_NCSI_MC_ID	0:0xFF		Byte 1 of the MCTP packet payload. Management Controller ID. In Control packets, this field identifies the Management Controller issuing the packet.
SMBus_NCSI_MC_ID			
VDM_NCSI_Header_Rev	0:0xFF		Byte 2 of the MCTP packet payload. This field identifies the version of the Control packet header in use by the sender.
SMBus_NCSI_Header_Rev			
VDM_NCSI_Reserved	0:0xFF		Byte 3 of the MCTP packet payload.
SMBus_NCSI_Reserved			

#### Example:

The following example creates MCTP event with **MCTP\_NC\_SI** type:

```

RecEvent = MCTP {
    MCTP_Bus = VDM_And_SMBus
    VDM_MCTPType = MCTP_NC_SI
    VDM_NCSI_MC_ID = "0x0a"
    SMBus_MCTPType = MCTP_NC_SI
    SMBus_NCSI_MC_ID = "0x0a"
}

```

#### 4.3.14.6 MCTPType = NVMe\_MI

Parameter	Values	Default	Comment
VDM_NVMe_MI_ROR	NVMe_MI_Request (0) NVMe_MI_Response (1) <i>or</i> 0:1		Bit 7 of byte 1 of the MCTP packet payload. This bit indicates whether the message is a Request Message or Response Message. This bit is cleared to '0' for Request Messages. This bit is set to '1' for Response Messages.
SMBus_NVMe_MI_ROR			
VDM_NVMe_MI_Type	NVMe_MI_ControlPrimitive (0x0) NVMe_MI_Command (0x1) NVMe_AdminCommand (0x2) NVMe_MI_Reserved (0x3) NVMe_MI_PCleCommand (0x4) <i>or</i> 0:0xF		Bits 6:3 of byte 1 of the MCTP packet payload. This field specifies the NVMe-MI Message Type.
SMBus_NVMe_MI_Type			
VDM_NVMe_MI_CSI	NVMe_MI_CommandSlot_0 (0) NVMe_MI_CommandSlot_1 (1) <i>or</i> 0:1		Bit 0 of byte 1 of the MCTP packet payload. This bit indicates the Command Slot with which the NVMe-MI Message is associated.
SMBus_NVMe_MI_CSI			
VDM_NVMe_MI_Reserved	0:0xFFFF		Bytes 2-3 of the MCTP packet payload.
SMBus_NVMe_MI_Reserved			

#### Example:

The following example creates MCTP event with **MCTP\_NVME\_MI** type:

```

RecEvent = MCTP {
    MCTP_Bus = VDM_And_SMBus
    VDM_MCTPType = MCTP_NVME_MI
    VDM_NVMe_MI_Type = NVMe_MI_Command
    VDM_NVMe_MI_CSI = NVMe_MI_CommandSlot_0
    SMBus_MCTPType = MCTP_NVME_MI
    SMBus_NVMe_MI_Type = NVMe_MI_Command
    SMBus_NVMe_MI_CSI = NVMe_MI_CommandSlot_0
}

```

### 4.3.15 RecEvent = Global\_Timer\_A, Global\_Timer\_B, Local\_Timer\_A, Local\_Timer\_B, Local\_Timer\_C, Local\_Timer\_D

This command creates Timer event in recording rules.

Two Global Timers (A and B) can be created in Global State and four Local Timers (A, B, C and D) can be created per each Local State.

Actions **Start\_Global\_Timer\_X / Start\_Local\_Timer\_X** and **Reset\_Global\_Timer\_X / Reset\_Local\_Timer\_X** can be used to start and reset timers.

#### Global\_Timer:

Global Timers are always created in Global State. They can be started and reset from events from any state.

#### Local\_Timer:

Local Timers can be created in any Local State. **CurrentState** parameter must be set to one of 16 Local States (**State\_1**, ..., **State\_16**). Local Timers can be started and reset from events only from the same Local State, where timer is located.

Parameter	Values	Default	Comment
Secs	0:12	0	Number of seconds.
Millisecs	0:999	0	Number of milliseconds.
Microsecs	0:999	0	Number of microseconds.
Nanosecs	0:999	0	Number of nanoseconds.

#### Example:

The following example creates Event and Global Timer:

```
RecEvent = Global_Timer_A {
    Action = Trigger
    Millisecs = 20
    Microsecs = 5
}
RecEvent = TLP {
    Action = Start_Global_Timer_A
}
```

The following example creates Event and Local Timer:

```
RecEvent = Local_Timer_A {
    CurrentState = State_1
    Action = Trigger
    Millisecs = 20
    Microsecs = 5
}
RecEvent = TLP {
    CurrentState = State_1
    Action = Start_Local_Timer_A
}
```

### 4.3.16 RecEvent = Global\_Counter\_1, Global\_Counter\_2, Local\_Counter\_1, Local\_Counter\_2, Local\_Counter\_3, Local\_Counter\_4

This command creates Counter event in recording rules.

**Channel** parameter must be set to **Upstream** or **Downstream** to create Counter event.

For each Channel two Global Counters (1 and 2) in Global State can be created and four Local Counters (1, 2, 3 and 4) can be created per each Local State.

Actions **Increment\_Global\_Counter\_X/ Increment\_Local\_Counter\_X** and **Reset\_Global\_Counter\_X/ Reset\_Local\_Counter\_X** can be used to increment and reset counters.

#### Global\_Counter:

Global Counters are always created in Global State. They can be incremented and reset from events from any state.

#### Local\_Counter:

Local Counters can be created in any Local State. **CurrentState** parameter must be set to one of 16 Local States (**State\_1, ..., State\_16**). Local Counters can be incremented and reset from events only from the same Local State, where timer is located.

Parameter	Values	Default	Comment
CounterValue	0:4294967295	2	Number of increments before performing Counter's actions.

#### Example:

The following example creates Event and Global Counter:

```
RecEvent = Global_Counter_1 {
    Channels = Upstream
    Action = Trigger
    CounterValue = 10
}
RecEvent = TLP {
    Channels = Upstream
    Action = Increment_Global_Counter_1
}
```

The following example creates Event and Local Counter:

```
RecEvent = Local_Counter_1
{
    Channels = Upstream
    CurrentState = State_1
    Action = Trigger
    CounterValue = 10
}
RecEvent = TLP
{
    Channels = Upstream
    CurrentState = State_1
    Action = Increment_Local_Counter_1
}
```

## 4.4 RecAction Command

### 4.4.1 RecAction=Jammer

#### 4.4.1.1 Common for all RecAction = Jammer fields

RecAction has the following properties, which are common for all types of action Jammer:

Parameter	Value	Default	Comment
ActionName	Custom string		It can be any custom string specified in quotation marks. This name should be used to associate the current action with <b>RecEvent</b> .
Count	0 : 4294967295	0	Number of jam repetition. <b>0</b> – means infinity
JammerType	JammerDelete JammerReplace JammerModify JammerInsertPktXAfter JammerInsertError JammerSidebandSignal		Supported types of Jammer Actions: <b>JammerDelete</b> – associated event will be deleted. <b>JammerReplace</b> – associated event will be replaced with specified another one. <b>JammerModify</b> – associated event will be modified with the data specified. <b>JammerInsertPktXAfter</b> – New specified event will be inserted after associated event. <b>JammerInsertError</b> – The one of supported error will be injected to associated event. <b>JammerSideBandSignal</b> – One of the available sideband signals will be generated after associated event.

#### 4.4.1.2 JammerType = JammerDelete

This jammer action will delete particular packet which is associated with corresponding event type.

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS deletion before any deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.

#### Example:

The following example shows how to delete 10 TS1 packets.

```

RecAction = Jammer
{
    ActionName = "JammerAction_0"
    Count = 10
    JammerType = JammerDelete
    EDSInsDelBeforeOS = Yes
    AlignNextLane0Pkt = Yes
}

RecEvent = TS1
{
    Channels = All
    CurrentState = State_Global
    Action = ( "JammerAction_0" )
}

```

#### 4.4.1.3 JammerType = JammerInsertPktXAfter, JammerReplace

Parameter	Value	Default	Comment
PacketType	Packet_L_IDLE Packet_TS1_PAD_PAD Packet_TS1_NUM_PAD Packet_TS1_NUM_NUM Packet_TS2_PAD_PAD Packet_TS2_NUM_NUM Packet_EDS Packet_EDB Packet_SKP Packet_CTRLSKP Packet_EIEOS Packet_EIOS Packet_FTS Packet_SDS Packet_DLLP Packet_TLP Packet_PAD		Supported packet types which can be inserted/replaced by jammer action.

The fields of JammerInsertPktXAfter/JammerReplace action depend on the type of packet which is inserted:

#### 4.4.1.3.1 Insert/Replace PacketType = Packet\_L\_IDLE

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
Packet Length	1:4		Length of packet in DWORDs

#### 4.4.1.3.2 Insert/Replace PacketType = Packet\_TS1\_PAD\_PAD, Packet\_TS1\_NUM\_PAD, Packet\_TS1\_NUM\_NUM, Packet\_TS2\_PAD\_PAD, Packet\_TS2\_NUM\_NUM, Packet\_EIEOS

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
G1G2_Raw_Data	4 DWs raw hex data	Default layout for specific packet	4 DWs length raw hex data which represents packet specific layout for PCIe Gen 1/ Gen 2.
G1G2_KBits_Data	4 x 4-bit binary nibbles	Default layout for specific packet	Each bit of the nibble corresponds to each byte of the dword in G1G2_Raw_Data. So the user are able to specify each byte of raw data as K-symbol.
G3G4_Raw_Data	4 DWs raw hex data	Default layout for specific packet	4 DWs length raw hex data which represents packet specific layout for PCIe Gen 3/ Gen 4.

## 4.4.1.3.3 Insert/Replace PacketType = Packet\_SKP, Packet\_EIOS, Packet\_FTS

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
G1G2_Raw_Data	1 DW raw hex data	Default layout for specific packet	1 DW length raw hex data which represents packet specific layout for PCIe Gen 1/ Gen 2.
G1G2_KBits_Data	4-bit binary nibble	Default layout for specific packet	Each bit of the nibble corresponds to each byte of the dword in G1G2_Raw_Data. So the user are able to specify each byte of raw data as K-symbol.
G3G4_Raw_Data	4 DWs raw hex data	Default layout for specific packet	4 DWs length raw hex data which represents packet specific layout for PCIe Gen 3/ Gen 4.

## 4.4.1.3.4 Insert/Replace PacketType = Packet\_EDS, Packet\_EDB

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
G3G4_Raw_Data	1 DW raw hex data	Default layout for specific packet	1 DW length raw hex data which represents packet specific layout for PCIe Gen 3/ Gen 4.

## 4.4.1.3.5 Insert/Replace PacketType = Packet\_CTRLSPK, Packet\_SDS

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
G3G4_Raw_Data	4 DWs raw hex data	Default layout for specific	4 DW length raw hex data which represents packet specific layout

		packet	for PCIe Gen 3/ Gen 4.
--	--	--------	------------------------

#### 4.4.1.3.6 Insert/Replace PacketType = Packet\_DLLP

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
Recompute_TlpLCRC_DllpCRC16	Yes, No	No	If <b>Yes</b> , CRC16 will be recomputed for the inserted/replaced packet.
G1G2_Raw_Data	2 DW raw hex data	Default layout for DLLP	1 DW length raw hex data which represents DLLP layout for PCIe Gen 1/ Gen 2.
G1G2_KBits_Data	2 * 4-bit binary nibble	Default layout for DLLP	Each bit of the nibble corresponds to each byte of the dword in G1G2_Raw_Data. So the user are able to specify each byte of raw data as K-symbol.
G3G4_Raw_Data	4 DWs raw hex data	Default layout for DLLP	2 DWs length raw hex data which represents DLLP layout for PCIe Gen 3/ Gen 4.

#### 4.4.1.3.7 Insert/Replace PacketType = Packet\_TLP

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
Recompute_TlpECRC	Yes, No	No	If <b>Yes</b> , ECRC will be recomputed for the inserted/replaced TLP.
Recompute_TlpSTPTokenCRC	Yes, No	No	If <b>Yes</b> , STP token and CRC will be recomputed for the target TLP.
SeqNumType	SameAsLastTLP NextAsLastTLP OutOfSeqTLP ValueFromRam	SameAsLastTLP	It is specify the type of sequence number for the target TLP. <b>SameAsLastTLP</b> – will use the same as last TLP passed on the bus. <b>NextAsLastTLPwill</b> – will use the next from the last TLP passed on the bus. <b>OutOfSeqTLP</b> – will use the sequence number which

			is out of current sequence TLP passed on the bus. <b>ValueFromRam</b> – will use the sequence number which is defined in Raw Data.
Packet_Length	5:64 Dwords		The target TLP Packet length. It should be at least 5 Dwords and less than 64. In case the TLP have Prefix it should be count in in this length.
Recompute_TlpLCRC_DllpCRC16	Yes, No	No	If <b>Yes</b> , LCRC will be recomputed for the inserted/replaced packet.
G1G2_Raw_Data	5:64 DW raw hex data		5:64 DW length raw hex data which represents TLP layout for PCIe Gen 1/ Gen 2.
G1G2_KBits_Data	5:64 * 4-bit binary nibble		Each bit of the nibble corresponds to each byte of the dword in G1G2_Raw_Data. So the user are able to specify each byte of raw data as K-symbol.
G3G4_Raw_Data	5:64 DWs raw hex data		5:64 DWs length raw hex data which represents TLP layout for PCIe Gen 3/ Gen 4.

#### 4.4.1.3.8 Insert/Replace PacketType = Packet\_PAD

Parameter	Value	Default	Comment
EDSInsDelBeforeOS	Yes, No	No	<b>Yes</b> means EDS insertion/deletion before any insertion/deletion of Ordered Set
Packet_Length	1:4		Length in dwords
AlignNextLane0Pkt	Yes, No	No	If <b>Yes</b> , next packet will be aligned to Lane 0.
G1G2_Raw_Data	1:4 DWs raw hex data		1:4 DWs length raw hex data which represents packet layout for PCIe Gen 1/ Gen 2.
G1G2_KBits_Data	1:4 * 4-bit binary nibble		Each bit of the nibble corresponds to each byte of the dword in G1G2_Raw_Data. So the user are able to specify each byte of raw data as K-symbol.
G3G4_Raw_Data	1:4 DWs raw hex data		1:4 DWs length raw hex data which represents packet layout for PCIe

			Gen 3/ Gen 4.
--	--	--	---------------

Example:

The following example shows how to replace any TLP with DLLP Ack 10 times.

```
RecAction = Jammer
{
    Count = 10
    ActionName = "JammerAction_0"
    JammerType = JammerReplace
    PacketType = Packet_DLLP
    EDSInsDelBeforeOS = Yes
    AlignNextLane0Pkt = Yes
    Recompute_TlpLCRC_DllpCRC16 = Yes
    G1G2_Raw_Data = ( "0x5c000000", "0x000000fd" )
    G1G2_KBits_Data = ( "0b1000", "0b0001" )
    G3G4_Raw_Data = ( "0xf0ac0000", "0x00000000" )
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    Action = ( "JammerAction_0" )
    TLP_Prefix = No
}
```

#### 4.4.1.4 JammerType = JammerModify

Parameter	Value	Default	Comment
SourceEventType	Skip CtrlSkip TS1 TS2 TLP DLLP Lane_Margining NVMe_Register NVMe_Command_Submission NVMe_Command_Completion		Supported packet types which can be modified by jammer action.

##### 4.4.1.4.1 Modify SourceEventType = Skip

Parameter	Value	Default	Comment
G1G2_Raw_Data	1 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	4 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 3/ Gen 4.

##### 4.4.1.4.2 Modify SourceEventType = TS1, TS2

Parameter	Value	Default	Comment
PacketType	For TS1: Packet_TS1_PAD_PAD Packet_TS1_NUM_PAD Packet_TS1_NUM_NUM  For TS2: Packet_TS2_PAD_PAD Packet_TS2_NUM_NUM		Particular TS type.
G1G2_Raw_Data	4 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	4 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 3/ Gen 4.

## 4.4.1.4.3 Modify SourceEventType = TLP

Parameter	Value	Default	Comment
Recompute_TlpECRC	Yes, No	No	If <b>Yes</b> , ECRC will be recomputed for the modified TLP.
Recompute_TlpLCRC_DllpCRC16	Yes, No	No	If <b>Yes</b> , LCRC will be recomputed for the modified packet.
Recompute_TlpSTPTokenCRC	Yes, No	No	If <b>Yes</b> , STP token and CRC will be recomputed for the target TLP.
G1G2_Modify_TLP_Prefix	1 DW raw hex data		Raw hex data which represents TLP prefix layout for PCIe Gen 1/ Gen 2.
G3G4_Modify_TLP_Prefix	1 DW raw hex data		Raw hex data which represents TLP prefix layout for PCIe Gen 3/ Gen 4
G1G2_Raw_Data	21 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	21 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 3/ Gen 4.

## 4.4.1.4.4 Modify SourceEventType = DLLP

Parameter	Value	Default	Comment
Recompute_TlpLCRC_DllpCRC16	Yes, No	No	If <b>Yes</b> , CRC16 will be recomputed for the modified packet.
G1G2_Raw_Data	2 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	2 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 3/ Gen 4.

## 4.4.1.4.5 Modify SourceEventType = CtrlSkip, Lane\_Margining

Parameter	Value	Default	Comment
G3G4_Raw_Data	4 DW raw hex data		Raw hex data which represents packet layout for PCIe Gen 3/ Gen 4.

## 4.4.1.4.6 Modify SourceEventType = NVMe\_Register

Parameter	Value	Default	Comment
Register_Data_Pattern	1 DW raw hex data		Raw hex data which represents NVMe register data pattern

## 4.4.1.4.7 Modify SourceEventType = NVMe\_Command\_Submission

Parameter	Value	Default	Comment
G1G2_Raw_Data	16 DW raw hex data		Raw hex data which represents NVMe submission command layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	16 DW raw hex data		Raw hex data which represents NVMe submission command for PCIe Gen 3/ Gen 4.

## 4.4.1.4.8 Modify SourceEventType = NVMe\_Command\_Completion

Parameter	Value	Default	Comment
G1G2_Raw_Data	4 DW raw hex data		Raw hex data which represents NVMe completion command layout for PCIe Gen 1/ Gen 2.
G3G4_Raw_Data	4 DW raw hex data		Raw hex data which represents NVMe completion command for PCIe Gen 3/ Gen 4.

Example:

The following example shows how to modify any TLP with Prefix 10 times.

```

RecAction = Jammer
{
    ActionName = "JammerAction_0"
    Count = 10
    JammerType = JammerModify
    SourceEventType = TLP
    G1G2_Modify_TLP_Prefix = "0x81XXXXXX"
    G3G4_Modify_TLP_Prefix = "0x81XXXXXX"
    G1G2_Raw_Data = (
        "0b0x0101XXXXXXXXXXXXXXXXXXXXXXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX"
    )
    G3G4_Raw_Data = (
        "0b0x0101XXXXXXXXXXXXXXXXXXXXXXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX",
        "0XXXXXXXX", "0XXXXXXXX", "0XXXXXXXX"
    )
}

```

```
        "0xFFFFFFFF", "0xFFFFFFFF", "0xFFFFFFFF", "0xFFFFFFFF",  
        "0xFFFFFFFF", "0xFFFFFFFF", "0xFFFFFFFF", "0xFFFFFFFF",  
        "0xFFFFFFFF", "0xFFFFFFFF", "0xFFFFFFFF"  
    )  
}  
RecEvent = TLP  
{  
    Channels = All  
    CurrentState = State_Global  
    Action = ( "JammerAction_0" )  
    TLP_Prefix = Yes  
    TLP_Prefix_Tag = MRIOV  
}
```

#### 4.4.1.5 JammerType = JammerInsertError

This jammer action type inserts error into particular packet in the traffic. Parameters and fields of this jammer action depend on error type and are described in details below.

##### 4.4.1.5.1 ErrorType = DROP\_STP\_SDP (can be applied to TLP/DLLP rec events)

This jammer error type removes STP or SDP symbol of TLP/DLLP and replaces it with other symbol.

Parameter	Values	Default	Comment
UseKbit	Yes, No		Indicates if First Symbol instead of STP/SDP should be K-Bit (applicable for Gen1/Gen2)
FirstDataSymbol	0x00:0xFF		Indicates value of symbol that will replace STP/SDP symbol
Recompute_TlpSTPTokenCRC	Yes, No		Indicates if it is necessary to re-compute STP/SDP token CRC and parity

#### Example:

The following example creates jammer actions that will Insert Error DROP\_STP\_SDP into TLP or DLLP packet.

```
RecAction = Jammer
{
    ActionName = "JammerAction_0"
    JammerType = JammerInsertError
    ErrorType = DROP_STP_SDP
    UseKbit = Yes
    FirstDataSymbol = 0x02
    Recompute_TlpSTPTokenCRC = No
}
```

#### 4.4.1.5.2 ErrorType = DROP\_END (can be applied to TLP/DLLP rec events)

This jammer error type removes END symbol of TLP/DLLP and replaces it with other symbol.

Parameter	Values	Default	Comment
UseKbit	Yes, No		Indicates if Last Symbol instead of END should be K-Bit (applicable for Gen1/Gen2)
LastDataSymbol	0x00:0xFF		Indicates value of symbol that will replace END symbol

#### Example:

The following example creates jammer actions that will Insert Error DROP\_END into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = DROP_END
    UseKbit = No
    LastDataSymbol = 0x01
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.3 ErrorType = NULLIFY\_TLP (can be applied to TLP rec events)

This type of error nullifies TLP. It doesn't have additional parameters.

Example:

The following example creates jammer actions that will Insert Error NULLIFY\_TLP into TLP packet.

```
RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = NULLIFY_TLP
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}
```

#### 4.4.1.5.4 ErrorType = STP\_LEN\_ERR (can be applied to TLP rec events)

This jammer error type corrupts length field of the TLP.

Parameter	Values	Default	Comment
ErrorFillValue	Increment Decrement All_0 All_1		Indicates incorrect value of TLP length "Increment" – length value is incremented "Decrement" – length value is decremented "All_0" – length value has all 0s bits "All_1" – length value has all 1s bits
Recompute_TlpSTPTokenCRC	Yes, No		Indicates if it is necessary to re-compute STP token CRC and parity

#### Example:

The following example creates jammer actions that will Insert Error NULLIFY\_TLP into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = STP_LEN_ERR
    ErrorFillValue = All_0
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.5 ErrorType = SEQ\_NUM\_ERR (can be applied to TLP rec events)

This jammer error type corrupts sequence number field of the TLP.

Parameter	Values	Default	Comment
ErrorFillValue	Increment Decrement OutOfSeq		Indicates incorrect value of TLP sequence number "Increment" – sequence number value is incremented "Decrement" – sequence number value is decremented "OutOfSeq" – sequence number value has value that is out of sequence
Recompute_TlpLCRC_DllpCRC16	Yes, No		Indicates if it is necessary to re-compute LCRC of the TLP

#### Example:

The following example creates jammer actions that will Insert Error SEQ\_NUM\_ERR into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = SEQ_NUM_ERR
    ErrorFillValue = OutOfSeq
    Recompute_TlpLCRC_DllpCRC16 = Yes
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.6 ErrorType = STP\_FCRC\_ERR (can be applied to TLP rec events)

This jammer error type corrupts FCRC of the TLP.

Parameter	Values	Default	Comment
ErrorFillValue	ToggleAllBits Corrupt_Bit_0 All_0 All_1		Indicates incorrect value of TLP FCRC "ToggleAllBits" – FCRC value has all bits incorrect "Corrupt_Bit_0" – FCRC value bit 0 is corrupted (incorrect) "All_0" – FCRC value has all 0s bits "All_1" – FCRC value has all 1s bits
Recompute_TlpSTPTokenCRC	Yes, No		Indicates if it is necessary to re-compute STP token CRC and parity

#### Example:

The following example creates jammer actions that will Insert Error STP\_FCRC\_ERR into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = STP_FCRC_ERR
    ErrorFillValue = ToggleAllBits
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.7 ErrorType = LCRC\_ERR (can be applied to TLP rec events)

This jammer error type corrupts LCRC field of the TLP.

Parameter	Values	Default	Comment
ErrorFillValue	ToggleAllBits Corrupt_Bit_0 All_0 All_1		Indicates incorrect value of TLP LCRC "ToggleAllBits" – LCRC value has all bits incorrect "Corrupt_Bit_0" – LCRC value bit 0 is corrupted (incorrect) "All_0" – LCRC value has all 0s bits "All_1" – LCRC value has all 1s bits

#### Example:

The following example creates jammer actions that will Insert Error LCRC\_ERR into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = LCRC_ERR
    ErrorFillValue = All_0
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.8 ErrorType = ECRC\_ERR (can be applied to TLP rec events)

This jammer error type corrupts ECRC of the TLP.

Parameter	Values	Default	Comment
ErrorFillValue	ToggleAllBits Corrupt_Bit_0 All_0 All_1		Indicates incorrect value of TLP ECRC "ToggleAllBits" – ECRC value has all bits incorrect "Corrupt_Bit_0" – ECRC value bit 0 is corrupted (incorrect) "All_0" – ECRC value has all 0s bits "All_1" – ECRC value has all 1s bits
Recompute_TlpLCRC_DllpCRC16	Yes, No		Indicates if it is necessary to re-compute LCRC of the TLP

#### Example:

The following example creates jammer actions that will Insert Error ECRC\_ERR into TLP packet.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = ECRC_ERR
    ErrorFillValue = All_1
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.5.9 ErrorType = DISPARITY\_ERR, SYMBOL\_ERR, SYNC\_BITS\_INV\_ERR, SYNC\_BITS\_ZERO\_ERR

This jammer error type inserts Disparity Error (DISPARITY\_ERR), Symbol Error (SYMBOL\_ERR), Sync Bits Inverted Error (SYNC\_BITS\_INV\_ERR) or Sync Bits Zero Error (SYNC\_BITS\_ZERO\_ERR) on specific lanes.

Parameter	Values	Default	Comment
ErrorByLane	( X, X, ..., X, X )		Array of 1 or 0 values that indicate on which lane error takes place (lanes from 0 to 15)

#### Example:

The following example creates jammer actions that will Insert Error DISPARITY\_ERR into TLP packet, on lanes 0, 1, 14, 15.

```

RecAction = Jammer
{
    ActionName = "JamExample"
    JammerType = JammerInsertError
    ErrorType = DISPARITY_ERR
    ErrorByLane = ( 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1 )
}

RecEvent = TLP
{
    Channels = All
    CurrentState = State_Global
    TLPTType = Mem
    Action = ( "JamExample" )
}

```

#### 4.4.1.6 JammerType = JammerSidebandSignal

This jammer action type causes Sideband Signal to be asserted/deasserted/bypassed in particular conditions.

Parameter	Values	Default	Comment
PEResetN	SB_Assert, SB_Deassert, SB_Bypass		Makes PEReset signal to be asserted/deasserted/bypassed
CLK_REQ	SB_Assert, SB_Deassert, SB_Bypass		Makes CLK_REQ signal to be asserted/deasserted/bypassed
WAKE_N	SB_Assert, SB_Deassert, SB_Bypass		Makes WAKE_N signal to be asserted/deasserted/bypassed
PWRBRK	SB_Assert, SB_Deassert, SB_Bypass		Makes PWRBRK signal to be asserted/deasserted/bypassed

#### Example:

The following example creates jammer actions that will assert Power Break signal:

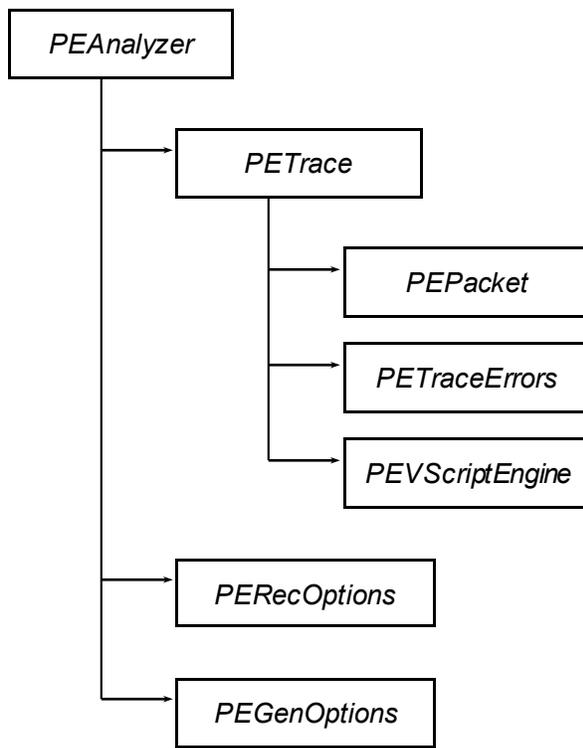
```
RecAction = Jammer
{
    ActionName = "JammerAction_0"
    JammerType = JammerSidebandSignal
    PWRBRK = SB_Assert
}
```

## 5 PETracer Object Model

Teledyne LeCroy's *PETracer*™ API programmatically exposes its functionality through objects. You work with an object by using its properties and methods. Objects are named according to the portion of an application they represent, and they are ordered in a hierarchy.

A single object occupies the topmost tier of Teledyne LeCroy *PETracer* object hierarchy: *PEAnalyzer*.

The following object model diagram shows how the objects in an object model fit together:



Only the *PEAnalyzer* object is creatable at the top level (for instance, via the *CoCreateInstance* call from a C/C++ client), instantiation of an object of other classes requires API calls.

The Class ID and App ID for the *PEAnalyzer* object are the following.

Class ID: 297CD804-08F5-4A4F-B3BA-779B2654B27C

App ID: CATC.PETracer

All interfaces are dual interfaces that allow simple use from typeless languages, like VBScript, as well as from C/C++.

All objects implement *ISupportErrorInfo* interface for easy error handling from the client.

<b>Objects</b>	<b>Interfaces</b>	<b>Description</b>
<i>PEAnalyzer</i>	<i>IAnalyzer</i> <i>IPEAnalyzer</i> <i>IPEAnalyzer2</i> <i>IPEAnalyzer3</i> <b><i>IPEAnalyzer5*</i></b> <i>_IAnalyzerEvents</i>	Represents the PCIe Protocol Analysis application
<i>PETrace</i>	<i>ITrace</i> <i>IPETrace</i> <i>IPETrace2</i> <i>IPETrace3</i> <i>IPEVerificationScript</i> <i>IPETraceForScript</i> <i>IPETraceForScript2</i> <i>IPETraceForScript3</i> <i>IPETraceForScript4</i> <i>IPETraceForScript5</i> <b><i>IPETraceForScript6*</i></b>	Represents recorded trace
<i>PERecOptions</i>	<i>IRecOptions</i> <i>IPERecOptions</i> <i>IPERecOptions2</i> <i>IPERecOptions3</i> <i>IPERecOptions4</i> <b><i>IPERecOptions5*</i></b>	Represents recording options
<i>PEGenOptions</i>	<i>IGenOptions</i> <i>IPEGenOptions</i> <i>IPEGenOptions2</i> <i>IPEGenOptions3</i> <b><i>IPEGenOptions4*</i></b>	Represents generation options
<i>PEPacket</i>	<i>IPacket</i> <b><i>IPEPacket*</i></b>	Represents single packet of the recorded trace
<i>AnalyzerErrors</i>	<b><i>IAnalyzerErrors*</i></b>	Represents the collection of errors occurred in the recorded trace

\* Primary interfaces

The examples of C++ code given in this document assume using the "import" technique of creating COM clients; that means the corresponding include is used:

```
#import "PEAutomation.tlb" no_namespace named_guids
```

Appropriate wrapper classes are created in .tli and .tlh files by the compiler.

*Remember to call CoInitialize or CoInitializeEx Windows API function in C++ code before the first Automation API object is created, and to call CoUninitialize after the last Automation API object is released (unless you are using some framework that does it for you). Failure to call CoInitialize or*

*CoInitializeEx would result in CoCreateInstance call failing with CO\_E\_NOTINITIALIZED error (HRESULT value of 0x800401F0).*

Samples of WSH, VBScript, and C++ client applications are provided.

## 6 PEAnalyzer Object

The *PEAnalyzer* object is a top-level object of *PETracer™* API.

The *PEAnalyzer* object allows user to control the recording and traffic generation, open trace files, and access to the recording and generation options.

The *PEAnalyzer* object supports the following interfaces:

Interfaces	Description
<i>IAnalyzer</i>	Facilitates recording and traffic generation, opens trace files, and retrieves recording options,
<i>IPEAnalyzer</i>	Extends the <i>IAnalyzer</i> interface: Adds advanced generator functionality, retrieves generation options.
<i>IPEAnalyzer2</i>	Extends the <i>IPEAnalyzer</i> interface: Adds hardware information and control methods.
<i>IPEAnalyzer3</i>	Extends the <i>IPEAnalyzer2</i> interface: Adds <i>StartImportFile</i> method.
<i>IPEAnalyzer5</i>	Extends the <i>IPEAnalyzer3</i> interface: - Adds <i>GetGenerationOptionsAsXML</i> method - Adds <i>SetGenerationOptionsAsXML</i> method
<i>IAnalyzerEvents</i>	Events from <i>PEAnalyzer</i> object

The *IPEAnalyzer5* interface is a primary interface for the *PEAnalyzer* object.

The Class ID and App ID for the *PEAnalyzer* object are the following.

Class ID: 297CD804-08F5-4A4F-B3BA-779B2654B27C  
App ID: CATC.PETracer

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
```

C++:

```
IPEAnalyzer* poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer5,
    (LPVOID *)&poPEAnalyzer ) )
    return;
```

## 6.1 IAnalyzer interface

The *IAnalyzer* interface is a dual interface for the *PEAnalyzer* object.

*IAnalyzer* implements the following methods:

- GetVersion*
- OpenFile*
- StartGeneration*
- StopGeneration*
- StartRecording*
- StopRecording*
- MakeRecording*
- LoadDisplayOptions*
- GetDisplayOptions*
- GetSerialNumber*

**Note:** All methods of the *IAnalyzer* interface are also available in the *IPEAnalyzer* (see Page 16), *IPEAnalyzer2* (see Page 20), *IPEAnalyzer3* (see Page ) and *IPEAnalyzer5* (see Page ) interfaces.

## 6.1.1 IAnalyzer::GetVersion

```
HRESULT GetVersion (
    [in] EAnalyzerVersionType version_type,
    [out, retval] WORD* analyzer_version )
```

Retrieves the current version of a specified subsystem.

### Parameters

version_type	Subsystem being queried for version; <i>EAnalyzerVersionType</i> enumerator has the following values: ANALYZERVERSION_SOFTWARE ( 0 ) – software
analyzer_version	Version of the subsystem queried

### Return values

ANALYZERCOMERROR_INVALIDVERSIONTYPE	Specified version type is invalid
-------------------------------------	-----------------------------------

### Remarks

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
SwVersion = Analyzer.GetVersion( 0 )
MsgBox "Software " & SwVersion
```

C++:

```
HRESULT hr;
IPEAnalyzer* poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer,
    (LPVOID *)&poPEAnalyzer ) )
    return;

WORD sw_version;
try
{
    sw_version = poAnalyzer->GetVersion( ANALYZERVERSION_SOFTWARE );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

TCHAR buffer[20];
_sprintf(buffer, _T("Software version:%X.%X"), HIBYTE(sw_version), LOBYTE(sw_version));
```

## 6.1.2 IAnalyzer::OpenFile

```
HRESULT OpenFile (
    [in] BSTR file_name,
    [out, retval] IDispatch** trace )
```

Opens a trace file, and creates the *PETrace* object.

### Parameters

file_name	String providing the full pathname to the trace file
trace	Address of a pointer to the <i>PETrace</i> object interface

### Return values

ANALYZERCOMERROR_UNABLEOPENFILE	Unable to open file
---------------------------------	---------------------

### Remarks

*PETrace* object is created via this method call, if the call was successful.

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.OpenFile( CurrentDir & "Input\errors.pex" )
```

C++:

```
HRESULT hr;
IPEAnalyzer* poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer,
    (LPVOID *)&poPEAnalyzer ) )
    return;

// open trace file
IDispatch* trace;
try
{
    trace = poPEAnalyzer->OpenFile( m_szRecFileName ).Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

// query for VTBL interface
IPETrace* pe_trace;
hr = trace->QueryInterface( IID_IPETrace, (LPVOID *)&pe_trace );
trace->Release();

if( FAILED(hr) )
    return;
```

### 6.1.3 IAnalyzer::StartGeneration

```
HRESULT StartGeneration (
    [in] BSTR gen_file_name,
    [in] long reserved1,
    [in] long reserved2 )
```

Starts traffic generation from the file.

#### Parameters

gen_file_name	String providing the full pathname to the generation file
reserved1	Reserved for future use
reserved2	Reserved for future use

#### Return values

ANALYZERCOMERROR_UNABLEOPENFILE	Unable to open file
ANALYZERCOMERROR_UNABLESTARTGENERATION	Unable to start generation (invalid state, etc.)

#### Remarks

#### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
ret = Analyzer.StartGeneration( CurrentDir & "Input\connect.peg", 0, 0 )
```

C++:

```
HRESULT          hr;
IPEAnalyzer*    poPEAnalyzer;
TCHAR           m_szGenFileName  [_MAX_PATH];

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer,
    (LPVOID *)&poPEAnalyzer ) )
    return;

. . .

try
{
    poAnalyzer->StartGeneration( m_szGenFileName, 0, 0 );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.1.4 IAnalyzer::StopGeneration

```
HRESULT StopGeneration ( )
```

Stops any current generation in progress.

### Return values

ANALYZERCOMERROR\_UNABLESTARTGENERATION      Unable to stop generation (invalid state, etc.)

### Remarks

### Example

C++:

```
IPEAnalyzer* poAnalyzer;  
  
. . .  
  
try  
{  
    poAnalyzer->StopGeneration();  
}  
catch ( _com_error& er)  
{  
    if (er.Description().length() > 0)  
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );  
    else  
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );  
    return 1;  
}
```

## 6.1.5 IAnalyzer::StartRecording

```
HRESULT StartRecording (
    [in] BSTR ro_file_name )
```

Starts recording with the specified recording options.

### Parameters

ro_file_name	String providing the full pathname to the recording options file; if the parameter is omitted, then recording starts with default recording options
--------------	---

### Return values

ANALYZERCOMERROR_UNABLESTARTRECORDING	Unable to start recording
---------------------------------------	---------------------------

### Remarks

After recording starts, this function returns. The analyzer continues recording until it is finished or until the *StopRecording* method call is performed. During the recording, the events are sent to event sink (see the *IAnalyzerEvents* interface, Page 119).

The recording options file is the file with extension *.rec* created by the PCIe Protocol Analysis application. You can create this file when you select "*Setup -> Recording Options...*" from the PCIe Protocol Analysis application menu, change the settings in the "*Recording Options*" dialog box, and then select the "*Save...*" button.

### Example

VBScript:

```
<OBJECT
    RUNAT=Server
    ID = Analyzer
    CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>

<INPUT TYPE=TEXT VALUE="" NAME="TextRecOptions">

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnStartRecording_OnClick
    On Error Resume Next
    Analyzer.StartRecording TextRecOptions.value
    If Err.Number <> 0 Then
        MsgBox Err.Number & ":" & Err.Description
    End If
End Sub
-->
</SCRIPT>
```

```
C++:
    IPEAnalyzer* pe_analyzer;
    BSTR         ro_file_name;

    . . .

    try
    {
        pe_analyzer->StartRecording( ro_file_name )
    }
    catch ( _com_error& er)
    {
        if (er.Description().length() > 0)
            ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
        else
            ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
        return 1;
    }
}
```

## 6.1.6 IAnalyzer::StopRecording

```
HRESULT StopRecording (
    [in] BOOL abort_upload )
```

Stops recording started by the *IAnalyzer::StartRecording* (see Page 10) method.

### Parameters

abort_upload	TRUE if the caller wants to abort the upload, no trace file is created; FALSE if the caller wants to upload the recorded trace
--------------	---

### Return values

ANALYZERCOMERROR_UNABLESTOPRECORDING	Error stopping recording
--------------------------------------	--------------------------

### Remarks

Stops recording that was started by the *StartRecording* method. The event is issued when recording is actually stopped (via the *\_IAnalyzerEvents* interface) if the parameter of this method call was FALSE.

### Example

VBScript:

```
<OBJECT
  RUNAT=Server
  ID = Analyzer
  CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnStopRecording_OnClick
  On Error Resume Next
  Analyzer.StopRecording True
  If Err.Number <> 0 Then
    MsgBox Err.Number & ":" & Err.Description
  End If
End Sub
-->
</SCRIPT>
```

C++:

```
IPEAnalyzer* pe_analyzer;

. . .

try
{
  pe_analyzer->StopRecording( FALSE )
}
catch ( _com_error& er)
{
  if (er.Description().length() > 0)
    ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
  else
    ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
  return 1;
}
```

## 6.1.7 IAnalyzer::MakeRecording

```
HRESULT MakeRecording (
    [in] BSTR ro_file_name,
    [out, retval] IDispatch** trace )
```

Makes recording with the specified recording options file.

### Parameters

ro_file_name	String providing the full pathname to a recording options file; if the parameter is omitted, then recording starts with default recording options
trace	Address of a pointer to the <i>PETrace</i> object interface

### Return values

ANALYZERCOMERROR_UNABLESTARTRECORDING	Unable to start recording
---------------------------------------	---------------------------

### Remarks

This method acts like the *StartRecording* method but does not return until recording is completed. The *PETrace* object is created via this method call if the call was successful.

The recording options file is the file with extension *.rec* created by the PCIe Protocol Analysis application. You can create this file when you select “*Setup -> Recording Options...*” from the PCIe Protocol Analysis application menu, change the settings in the “*Recording Options*” dialog box, and then select the “*Save...*” button.

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
```

C++:

```
IDispatch* trace;
IPEAnalyzer* pe_analyzer;
BSTR ro_file_name;
HRESULT hr;

. . .

try
{
    trace = pe_analyzer->MakeRecording( ro_file_name ).Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

// query for VTBL interface
IPETrace* pe_trace;
hr = trace->QueryInterface( IID_IPETrace, (LPVOID *)&pe_trace );
trace->Release();
```

## 6.1.8 IAnalyzer::LoadDisplayOptions

```
HRESULT LoadDisplayOptions (  
    [in] BSTR do_file_name )
```

Loads display options that apply to a trace opened or recorded later.

### Parameters

`do_file_name` String providing the full pathname to a display options file

### Return values

`ANALYZERCOMERROR_UNABLELOADDO` Unable to load the display options file

### Remarks

Use this method if you want to filter traffic of some type. The display options loaded by this method call apply only on trace file opened or recorded after this call.

Display options file is the file with extension `.opt` created by the PCIe Protocol Analysis application. You can create this file when you select “*Setup -> Display Options...*” from the PCIe Protocol Analysis application menu, change the settings in the “Display Options” dialog box, and then select the “Save...” button.

### Example

See *ITrace::ApplyDisplayOptions*, Page 27.

## 6.1.9 IAnalyzer::GetRecordingOptions

```
HRESULT GetRecordingOptions (
    [out, retval] IDispatch** recording_options )
```

Retrieves the interface for access to the recording options.

### Parameters

recording\_options                      Address of a pointer to the *PERecOptions* object interface

### Return values

### Remarks

*PERecOptions* object is created via this method call, if the call was successful.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
```

C++:

```
HRESULT            hr;
IPEAnalyzer*      poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer,
    (LPVOID *)&poPEAnalyzer ) )
    return;

// open trace file
IDispatch* rec_opt;
try
{
    rec_opt = poPEAnalyzer->GetRecordingOptions().Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

// query for VTBL interface
IPERecOptions* ib_rec_opt;
hr = rec_opt->QueryInterface( IID_IPERecOptions, (LPVOID *)&ib_rec_opt );
rec_opt->Release();

if( FAILED(hr) )
    return;
```

## 6.1.10 IAnalyzer::GetSerialNumber

```
HRESULT GetSerialNumber ( [out, retval] WORD* serial_number )
```

Returns analyzer's serial number.

### Parameters

### Return values

WORD serial\_number            Serial number of analyzer

### Remarks

### Example

WSH:

```
Set Analyzer = WScript.CreateObject("CATC.PETracer")
SerialNum = Analyzer.GetSerialNumber()
WScript.Echo "SerialNum= " & CLng(SerialNum)
```

C++:

```
HRESULT            hr;
IPEAnalyzer*      poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
          CLSID_PEAnalyzer,
          NULL, CLSCTX_SERVER,
          IID_IPEAnalyzer,
          (LPVOID *)&poPEAnalyzer ) )
    return;

WORD serial_number(0);
try
{
    serial_number = poAnalyzer->GetSerialNumber();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

TCHAR buffer[20];
_sprintf(buffer, _T("Serial number: %u"), serial_number );
```

## 6.2 IPEAnalyzer interface

The *IPEAnalyzer* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IAnalyzer* interface.

The *IPEAnalyzer* interface implements all methods from *IAnalyzer* interface plus the following:

*GetGenerationOptions*

*ResumeGeneration*

*GetLink Status*

**Note:** All methods implemented by the *IPEAnalyzer* interface are also implemented by the *IPEAnalyzer2*, *IPEAnalyzer3* and *IPEAnalyzer5* interfaces (see Page 20).

## 6.2.1 IPEAnalyzer::GetGenerationOptions

```
HRESULT GetGenerationOptions (
    [out, retval] IDispatch** generation_options )
```

Retrieves the interface for access to the generation options.

### Parameters

generation\_options                      Address of a pointer to the *PEGenOptions* object interface

### Return values

### Remarks

*PEGenOptions* object is created via this method call, if the call was successful.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions( )
```

C++:

```
HRESULT            hr;
IPEAnalyzer*      poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
    CLSID_PEAnalyzer,
    NULL, CLSCTX_SERVER,
    IID_IPEAnalyzer,
    (LPVOID *)&poPEAnalyzer ) )
    return;

// open trace file
IDispatch* gen_opt;
try
{
    gen_opt = poPEAnalyzer->GetGenerationOptions().Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

// query for VTBL interface
IPEGenOptions* pe_gen_opt;
hr = gen_opt->QueryInterface( IID_IPEGenOptions, (LPVOID *)&pe_gen_opt );
gen_opt->Release();

if( FAILED(hr) )
    return;
```

## 6.2.2 IPEAnalyzer::ResumeGeneration

```
HRESULT ResumeGeneration ( )
```

Resumes generation if it was previously paused.

### Return values

### Remarks

### Example

```
C++:  
IPEAnalyzer* poAnalyzer;  
  
. . .  
  
try  
{  
    poAnalyzer->ResumeGeneration();  
}  
catch ( _com_error& er)  
{  
    if (er.Description().length() > 0)  
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );  
    else  
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );  
    return 1;  
}
```

## 6.2.3 IPEAnalyzer::GetLinkStatus

```
HRESULT GetLinkStatus (
    [out] VARIANT* fc_status ,
    [out, retval] BSTR* link_status )
```

Returns two text strings with the link and flow control status.

### Parameters

fc_status	Flow control status, one of the following values can be returned: "Pending", "Complete", or "Not initialized"
link_status	Link status, one of the following values can be returned: "Detect. Quiet", "Detect. Active", "Polling.Active", "Polling.Compliance", "Polling.Configuration", "Polling.Speed", "Configuration.Linkwidth.Start", "Configuration.Linkwidth.Accept", "Configuration.Lanenum.Wait", "Configuration.Lanenum.Accept", "Configuration.Complete", "Configuration.Idle", "L0", "L0s.Idle", "L0s.FTS", "L1", "L2", "Recovery.RcvrLock", "Recovery.RcvrCfg", "Recovery.Idle", "Loopback", "Hot Reset", or "Disabled"

### Return values

### Remarks

### Example

```
C++:
    IPEAnalyzer* poAnalyzer;

    . . .

    BSTR link_status;           // Link Status
    VARIANT pe_status;        // Flow Control
    VariantInit(&pe_status);
    try
    {
        link_status = poAnalyzer->GetLinkStatus( &pe_status );
    }
    catch ( _com_error& er )
    {
        if (er.Description().length() > 0)
            ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
        else
            ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
        return 1;
    }

    USES_CONVERSION;
    TCHAR str_status[512];
    _tcscpy( str_status, "Link Status: ");
    _tcscat( str_status, OLE2T( link_status ) );
    _tcscat( str_status, "; Flow Control: ");
    _tcscat( str_status, OLE2T(V_BSTR(&pe_status)) );

    SysFreeString( link_status );

    ::MessageBox( NULL, str_status, _T("Status"), MB_OK );
```

### 6.2.4 IPEAnalyzer::GetLinkConfigEx

This method returns the linkwidth and speed of a connected analyzer or exerciser. Following is the method signature:

```
HRESULT GetLinkConfigEx ( [in] EHardwareType version_type,
                          [out] int* link_speed,
                          [out] int* link_width )
```

EHardwareType is defined as

```
HARDWARETYPE_PETRACER    = 0, for analyzer
HARDWARETYPE_PETRAINER   = 1, for exerciser
```

User should pass one of this value to get the correct link speed and width

Following are the return values;

For Analyzer i.e. when user pass HARDWARETYPE\_PETRACER (or 0)

Return ID	Linkwidth
-1	No link
1	x1
2	x2
4	x4
8	x8
16	x16

For Exerciser i.e. when user pass HARDWARETYPE\_PETRAINER (or 1)

Return ID	Linkwidth
0	No link
1	x1
2	x2
3	x4
4	x8
6	x16

Link speed (same values for Analyzer and Exerciser)

Return ID	Link speed
1	2.5 GT/s
2	5.0 GT/s
3	8.0 GT/s
4	16.0 GT/s

#### Example

An example /GetLinkwidthSpeed.py/ in python is available here  
 C:\Users\Public\Documents\LeCroy\PCle Protocol Suite\Automation\python\

## 6.3 IPEAnalyzer2 interface

The *IPEAnalyzer2* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer* interface.

The *IPEAnalyzer2* interface implements all methods from *IPEAnalyzer* interface plus the following:

- GetHardwareInfo*
- ResetHardware*

### 6.3.1 IPEAnalyzer2::GetHardwareInfo

```
HRESULT GetHardwareInfo (
    [in] EHardwareType type,
    [out, retval] int* info )
```

Returns information about the hardware (Summit Analyzer or Exerciser) connected.

#### Parameters

type	Hardware type being queried; the <i>EHardwareType</i> enumerator has the following values: HARDWARETYPE_PETRACER ( 0 ) - Summit Analyzer HARDWARETYPE_PETRAINER ( 1 ) - Summit Exerciser
------	--

info	The following values can be returned When type is HARDWARETYPE_PETRACER: 1 – (reserved) 2 – (reserved) 3 – (reserved) 4 – (reserved) 5 – (reserved) 6 – (reserved) 7 – Summit T3-16 8 – Summit T3-8 9 – Summit T28 10 – Summit T3-8 (2 units) 11 – Summit T24 12 – Summit T34 13 – Summit T34 (2 units) 14 – Eclipse X34 15 – Eclipse X34 (2 units) 16 – Summit Z416 17 – Summit T416 18 – Summit T48 19 – Summit Z4-1 PTC 20 – Summit M5x
------	---

When type is HARDWARETYPE\_PETRAINER:  
 0 – (reserved)  
 1 – (reserved)  
 2 – (reserved)  
 3 – Summit Z3-16  
 4 – Summit Z416

## Return values

## Remarks

## Example

C++:

```
IPEAnalyzer2* poAnalyzer;

. . .

int tracer_type = 0;
int trainer_type = 0;
try
{
    trainer_type = poAnalyzer->GetHardwareInfo( HARDWARETYPE_PETRAINER );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

try
{
    tracer_type = poAnalyzer->GetHardwareInfo( HARDWARETYPE_PETRACER );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

### 6.3.2 IPEAnalyzer2::ResetHardware

```
HRESULT ResetHardware (
    [in] EHardwareType type,
    [in] EResetType reset_type )
```

Resets the hardware specified.

#### Parameters

type	Hardware type to reset; the <i>EHardwareType</i> enumerator has the following values: HARDWARETYPE_PETRACER      ( 0 ) - Summit Analyzer HARDWARETYPE_PETRAINER    ( 1 ) - Summit Exerciser
reset_type	Type of the reset; the <i>EResetType</i> enumerator has the following values: RESETPETYPE_LINK           ( 0 ) - link reset

#### Remarks

#### Example

C++:

```
IPEAnalyzer2* poAnalyzer;
. . .

try
{
    poAnalyzer->ResetHardware( HARDWARETYPE_PETRAINER, RESETPETYPE_LINK );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.4 IPEAnalyzer3 interface

The *IPEAnalyzer3* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer2* interface.

The *IPEAnalyzer3* interface implements all methods from *IPEAnalyzer2* interface plus the following:  
*StartImportFile*

### 6.4.1 IPEAnalyzer3::StartImportFile

```
HRESULT StartImportFile ( [in] BSTR file_name )
```

Begins process of importing specified file into a trace file.

#### Parameters

file name	String providing the full pathname to the import file
-----------	---

#### Return values

S_OK	Import started successfully
S_FALSE	Error starting import

#### Remarks

The format of the import file is defined in a proprietary Teledyne LeCroy specification. Please contact Teledyne LeCroy PSG for details.

#### Example

C++:

```
IPEAnalyzer3* poAnalyzer;

. . .

HRESULT hr;
try
{
    hr = poAnalyzer->StartImportFile( szFileName );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.5 IPEAnalyzer5 interface

The *IPEAnalyzer5* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer3* interface.

The *IPEAnalyzer5* interface implements all methods from *IPEAnalyzer3* interface plus the following:

- GetGenerationOptionsAsXML*
- SetGenerationOptionsFromXML*

## 6.5.1 IPEAnalyzer5::GetGenerationOptionsAsXML

```
HRESULT GetGenerationOptionsAsXML( [in] BSTR xml_file_path )
```

Exports generation options to XML file.

### Parameters

xml_file_path	String providing the full pathname to the export XML file
---------------	---

### Return values

S_OK	Exported successfully
S_FALSE	Error starting export

### Remarks

### Example

C++:

```
IPEAnalyzer5* poAnalyzer;

. . .

HRESULT hr;
try
{
    hr = poAnalyzer->GetGenerationOptionsAsXML( szFileName );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.5.2 IPEAnalyzer5::SetGenerationOptionsFromXML

```
HRESULT SetGenerationOptionsFromXML( [in] BSTR xml_file_path )
```

Imports generation options from XML file.

### Parameters

xml_file_path	String providing the full pathname to the import XML file
---------------	---

### Return values

S_OK	Imported successfully
S_FALSE	Error starting import

### Remarks

### Example

C++:

```
IPEAnalyzer5* poAnalyzer;  
  
. . .  
  
HRESULT hr;  
try  
{  
    hr = poAnalyzer->SetGenerationOptionsFromXML( szFileName );  
}  
catch ( _com_error& er)  
{  
    if (er.Description().length() > 0)  
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );  
    else  
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );  
    return 1;  
}
```

## 6.6 IPEAnalyzer6 interface

The *IPEAnalyzer6* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer5* interface.

The *IPEAnalyzer6* interface implements all methods from *IPEAnalyzer5* interface plus the following:

- GetRecordingOptionsAsXML*
- SetRecordingOptionsFromXML*

## 6.6.1 IPEAnalyzer6::GetRecordingOptionsAsXML

```
HRESULT GetRecordingOptionsAsXML ( [in] BSTR xml_file_path )
```

Exports recording options to XML file.

### Parameters

xml_file_path	String providing the full pathname to the export XML file
---------------	---

### Return values

S_OK	Exported successfully
S_FALSE	Error starting export

### Remarks

### Example

C++:

```
IPEAnalyzer6* poAnalyzer;

. . .

HRESULT hr;
try
{
    hr = poAnalyzer->GetRecordingOptionsAsXML( szFileName );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.6.2 IPEAnalyzer6::SetRecordingOptionsFromXML

```
HRESULT SetRecordingOptionsFromXML( [in] BSTR xml_file_path )
```

Imports recording options from XML file.

### Parameters

xml_file_path	String providing the full pathname to the import XML file
---------------	---

### Return values

S_OK	Imported successfully
S_FALSE	Error starting import

### Remarks

### Example

C++:

```
IPEAnalyzer6* poAnalyzer;  
  
. . .  
  
HRESULT hr;  
try  
{  
    hr = poAnalyzer->SetRecordingOptionsFromXML( szFileName );  
}  
catch ( _com_error& er)  
{  
    if (er.Description().length() > 0)  
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );  
    else  
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );  
    return 1;  
}
```

## 6.7 IPEAnalyzer8 interface

The *IPEAnalyzer8* interface is a primary dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer7* interface.

The *IPEAnalyzer8* interface implements all methods from *IPEAnalyzer7* interface plus the following:  
*GetSerialNumberEx*

## 6.7.1 IPEAnalyzer8::GetSerialNumberEx

```
HRESULT GetSerialNumber( [in] EHardwareType version_type,
                        [out, retval] WORD* serial_number )
```

Returns either analyzer's or exerciser's serial number depending on input parameter.

### Parameters

EHardwareType version\_type    HARDWARETYPE\_PETRACER (0) for analyzer,  
                                   HARDWARETYPE\_PETRAINER (1) for exerciser

### Return values

WORD serial\_number                Serial number of either analyzer or exerciser

### Remarks

### Example

WSH:

```
Set Analyzer = WScript.CreateObject("CATC.PETTracer")
TracerSerialNum = Analyzer.GetSerialNumberEx(0)
TrainerSerialNum = Analyzer.GetSerialNumberEx(1)
WScript.Echo "AnalyzerSerialNum= " & CLng(TracerSerialNum)
WScript.Echo "ExerciserSerialNum= " & CLng(TrainerSerialNum)
```

C++:

```
HRESULT            hr;
IPEAnalyzer8*     poPEAnalyzer;

// create PEAnalyzer object
if ( FAILED( CoCreateInstance(
          CLSID_PEAnalyzer,
          NULL, CLSCTX_SERVER,
          IID_IPEAnalyzer8,
          (LPVOID *)&poPEAnalyzer ) )
    return;

WORD tracer_serial_number(0);
WORD trainer_serial_number(0);
try
{
    tracer_serial_number = poAnalyzer->GetSerialNumberEx(0);
    trainer_serial_number = poAnalyzer->GetSerialNumberEx(1);
}
catch (_com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETTracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETTracer client"), MB_OK );
    return 1;
}

TCHAR buffer[60];
_sprintf(buffer, _T("Analyzer serial number: %u, Exerciser serial number: %u"),
tracer_serial_number, trainer_serial_number );
```

## 6.8 IPEAnalyzer23::SetDeviceRecordingEnabled

```
HRESULT SetDeviceRecordingEnable( [in] BSTR serial_number,  
                                 [in] BOOL enabled);
```

### Description

Enable/Disable recording using Analyzer with given serial number.

### Parameters

serial\_number – analyzer serial number, enabled – enable flag.

### Example

Please check C:\Users\Public\Documents\LeCroy\PCle Protocol Suite\Automation\python\disable\_z4.py

## 6.9 IPEAnalyzer28 interface

The *IPEAnalyzer28* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer27* interface.

The *IPEAnalyzer28* interface implements all methods from *IPEAnalyzer27* interface plus the following:

*StartJammingAndRecording*

*StopJammingAndRecording*

*StartJamming*

*StopJamming*

### 6.9.1 IPEAnalyzer28::StartJammingAndRecording

```
HRESULT StartJammingAndRecording( [in] BSTR ro_file_name );
```

#### Description

Start jamming and recording with specified recording options, if recording options are omitted then recording starts with default recording options.

#### Parameters

`ro_file_name` String that contains pathname of the recording options file. String "DO\_NOT\_CHANGE\_RECORDING\_OPTIONS" can be passed to use default recording options .

## 6.9.2 IPEAnalyzer28::StopJammingAndRecording

```
HRESULT StopJammingAndRecording( [in] BOOL abort_upload );
```

### Description

Stop jamming and recording.

### Parameters

abort_upload	TRUE if uploading must be aborted.
--------------	------------------------------------

### 6.9.3 IPEAnalyzer28::StartJamming

```
HRESULT StartJamming( [in] BSTR ro_file_name );
```

#### Description

Start jamming with specified recording options, if recording options are omitted then jamming starts with default recording options.

#### Parameters

`ro_file_name` String that contains pathname of the recording options file. String "DO\_NOT\_CHANGE\_RECORDING\_OPTIONS" can be passed to use default recording options .

## 6.9.4 IPEAnalyzer28::StopJamming

```
HRESULT StopJamming();
```

### Description

Stop jamming.

## 6.10 IPEAnalyzer31 interface

The *IPEAnalyzer31* interface is a dual interface for the *PEAnalyzer* object.

This interface is derived from the *IPEAnalyzer30* interface.

The *IPEAnalyzer31* interface implements all methods from *IPEAnalyzer30* interface plus the following:  
*SwitchM5XRecordingType*

### 6.10.1 IPEAnalyzer31::SwitchM5XRecordingType

```
HRESULT SwitchM5XRecordingType ([in] ERecordingType type);
```

#### Description

Switch recording type for M5X. If the type is Unknown, it is applied from default recording options.

#### Parameters

type	Recording type being queried; the <i>ERecordingType</i> enumerator can have the following values in call of this method: REC_TYPE_M5X ( 23 ) – Analyzer-Jammer mode REC_TYPE_T58 ( 25 ) – Analyzer mode REC_TYPE_M5X_AJA ( 28 ) – Analyzer-Jammer-Analyzer mode REC_TYPE_UNKNOWN ( 0xFFFF ) – Unknown mode
------	--

# PETrace Object

*PETrace* object represents the recorded trace file.

The *PETrace* object allows user to:

- Get trace information
- Access trace packets
- Access trace errors
- Save/export the trace or a portion of the trace

The *PETrace* object can be created by:

- Using *IAnalyzer::OpenFile* method (see Page **Error! Bookmark not defined.**)
- Using *IAnalyzer::MakeRecording* method (see Page 13)
- Handling *\_IAnalyzerEvents::OnTraceCreated* event (see Page 120)

The *PETrace* object supports the following interfaces:

Interfaces	Description
<i>ITrace</i>	Implements trace packets and trace errors access, different report types, export, and saving.
<i>IPETrace</i>	Extends <i>ITrace</i> interface: Adds the functionality for accessing the <i>PETracePacket</i> object.
<i>IPEVerificationScript</i>	Exposes the functionality for running verification scripts

The *IPETrace* interface is a primary interface for the *PETrace* object.

## 6.11 ITrace interface

The *ITrace* interface is a dual interface for the *PETrace* object.

It implements the following methods:

- GetName*
- ApplyDisplayOptions*
- Save*
- ExportToText*
- Close*
- ReportFileInfo*
- ReportErrorSummary*
- ReportTrafficSummary*
- GetPacket*
- GetPacketsCount*
- GetTriggerPacketNum*
- AnalyzerErrors*

**Note:** All methods of *ITrace* interface are also available in *IPETrace* (see Page 41).

### 6.11.1 ITrace::GetName

```
HRESULT GetName (
    [out, retval] BSTR* trace_name )
```

Retrieves the trace name.

#### Parameters

trace\_name                      Name of the trace

#### Return values

#### Remarks

This name can be used for presentation purposes.  
Do not forget to free the string returned by this method call.

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETTracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
MsgBox "Trace name " & Trace.GetName
```

C++:

```
IPETTrace* pe_trace;

. . .

_bstr_t bstr_trace_name;
try
{
    bstr_trace_name = pe_trace->GetName();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETTracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETTracer client"), MB_OK );
    return 1;
}

TCHAR str_trace_name[256];
_tcscpy( str_trace_name, (TCHAR*)( bstr_trace_name ) );
SysFreeString( bstr_trace_name );

::MessageBox( NULL, str_trace_name, _T("Trace name"), MB_OK );
```

## 6.11.2 ITrace::ApplyDisplayOptions

```
HRESULT ApplyDisplayOptions (
    [in] BSTR do_file_name )
```

Applies the specified display options to the trace.

### Parameters

`do_file_name`                      String providing the full pathname to the display options file

### Return values

`ANALYZERCOMERROR_UNABLELOADDO`                      Unable to load the display options file

### Remarks

Use this method if you want to filter traffic of some type in the recorded or opened trace.

The display options file is the file with extension `.opt` created by the PCIe Protocol Analysis™ application. You can create this file when you select “*Setup -> Display Options...*” from the PCIe Protocol Analysis application menu, change the settings in the “Display Options” dialog box, and then select the “Save...” button.

**Note:** This does not work on Multisegment traces

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
Trace.ApplyDisplayOptions      CurrentDir & "Input\test_do.opt"
Trace.Save                      CurrentDir & "Output\saved_file.pex"
```

C++:

```
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];

. . .

try
{
    pe_trace->ApplyDisplayOptions( file_name );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

### 6.11.3 ITrace::Save

```
HRESULT Save (
    [in] BSTR file_name,
    [in, defaultvalue(-1)] long packet_from,
    [in, defaultvalue(-1)] long packet_to )
```

Saves trace into a file while allowing you to specify a range of packets.

#### Parameters

file_name	String providing the full pathname to file where the trace is saved
packet_from	<i>beginning packet number</i> when you are saving a range of packets; value -1 means that the first packet of the saved trace is the first packet of this trace
packet_to	<i>ending packet number</i> when you are saving a range of packets; value -1 means that the last packet of the saved trace is the last packet of this trace

#### Return values

ANALYZERCOMERROR_UNABLESAVE	Unable to save the trace file
ANALYZERCOMERROR_INVALIDPACKETNUMBER	Bad packet range

#### Remarks

Use this method if you want to save a recorded or an opened trace into a file. If the display options applied to this trace (see *ITrace::ApplyDisplayOptions* on Page 27 or *IAnalyzer::LoadDisplayOptions* on Page 14), then hidden packets would not be saved.

If the packet range specified is invalid (for example, *packet\_to* is more than the last packet number in the trace, or *packet\_from* is less than the first packet number in the trace, or *packet\_from* is more than *packet\_to*), then the packet range is adjusted automatically.

#### Example

```
WSH:
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
Trace.ApplyDisplayOptions CurrentDir & "Input\test_do.opt"
Trace.Save CurrentDir & "Output\saved_file.pex"

C++:
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];
LONG packet_from;
LONG packet_to;
. . .
try
{
    pe_trace->Save( file_name, packet_from, packet_to );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

### 6.11.4 ITrace::ExportToText

```
HRESULT ExportToText (
    [in] BSTR file_name,
    [in, defaultvalue(-1)] long packet_from,
    [in, defaultvalue(-1)] long packet_to );
```

Exports the trace into a text file while allowing you to specify a range of packets.

#### Parameters

file_name	String providing the full file pathname for the exported trace
packet_from	<i>beginning packet number</i> when you are exporting a range of packets; value -1 means that the first packet of the exported trace is the first packet of this trace
packet_to	<i>ending packet number</i> when you are exporting a range of packets, value -1 means that the last packet of the exported trace is the last packet of this trace

#### Return values

ANALYZERCOMERROR_UNABLESAVE	Unable to export trace file
-----------------------------	-----------------------------

#### Remarks

Use this method if you want to export a recorded or an opened trace into a text file. If the display options applied to this trace (see *ITrace::ApplyDisplayOptions* on Page 27 or *IAnalyzer::LoadDisplayOptions* on Page 14), then hidden packets would not be exported.

If the packet range is specified and it is invalid (for example, *packet\_to* is more than the last packet number in the trace, or *packet\_from* is less than the first packet number in the trace, or *packet\_from* is more than *packet\_to*), then packet range is adjusted automatically.

## Here is a snippet of an exported text file:

File C:\data.pex.

From Packet #1880 to Packet #1890.

```
Packet#
-----|
Packet(1880) Downstream DLLP ACK AckNak_Seq_Num(3388) CRC 16(0xBB63)
-----|
Time Stamp(0002 . 069 437 652 s)
-----|
Packet(1881) Upstream SKIP COM(K28.5 ) SKIP Symbols(K28.0 K28.0 K28.0 )
-----|
Time Stamp(0002 . 069 437 848 s)
-----|
Packet(1882) Upstream DLLP UpdateFC-P VC ID(0) HdrFC(1) DataFC(2)
-----|
CRC 16(0x6744) Time Stamp(0002 . 069 437 936 s)
-----|
Packet(1883) Upstream DLLP UpdateFC-NP VC ID(0) HdrFC(1) DataFC(2)
-----|
CRC 16(0x8C23) Time Stamp(0002 . 069 437 944 s)
-----|
Packet(1884) Upstream DLLP UpdateFC-Cpl VC ID(0) HdrFC(6) DataFC(1287)
-----|
CRC 16(0x06F2) Time Stamp(0002 . 069 437 952 s)
-----|
Packet(1885) Downstream Packet Error(CodeErr, DlmtErr, LCRCErr) TLP(1285)
-----|
Cpl CplD(10:01010) RequesterID(058:22:1) Tag(177)
-----|
CompleterID(000:07:2) Status(UR)-BAD BCM(1) Byte Cnt(2618)-BAD
-----|
Lwr Addr(0x31)-BAD LCRC(0xB1000000)-BAD
-----|
Time Stamp(0002 . 069 437 956 s)
-----|
Packet(1886) Upstream TLP(3389) Cfg CfgRd0(00:00100) RequesterID(000:00:0)
-----|
Tag(0) DeviceID(000:00:0) Register(0x000) 1st BE(0000) LCRC(0xF1AB6932)
-----|
Time Stamp(0002 . 069 437 960 s)
-----|
Packet(1887) Downstream TS2 COM(K28.5 ) Link Lane N_FTS
-----|
Training Control TS2 Time Stamp(0002 . 069 437 976 s)
-----|
Packet(1888) Upstream DLLP Vendor Data(01 02 03) CRC 16(0x532D)
-----|
Time Stamp(0002 . 069 437 984 s)
-----|
Packet(1889) Downstream TS2 COM(K28.5 ) Link Lane N_FTS
-----|
Training Control TS2 Time Stamp(0002 . 069 438 040 s)
-----|
Packet(1890) Upstream SKIP COM(K28.5 ) SKIP Symbols(K28.0 K28.0 K28.0 )
-----|
Time Stamp(0002 . 069 438 072 s)
-----|
```

## Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
Trace.ApplyDisplayOptions    CurrentDir & "Input\test_do.opt"
Trace.ExportToText          CurrentDir & "Output\text_export.txt"
```

C++:

```
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];
LONG packet_from;
LONG packet_to;
. . .
try
{
    pe_trace->ExportToText( file_name, packet_from, packet_to );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

### 6.11.5 ITrace::Close

```
HRESULT Close ( )
```

Closes the trace.

#### Parameters

#### Return values

#### Remarks

Closes the current trace, but does not release the interface pointer. Call *IUnknown::Release* method right after this method call. No *ITrace* method call succeeds after calling *ITrace::Close* method. (Currently, there is no need to call *ITrace::Close* directly since *IUnknown::Release* closes the trace.)

#### Example

## 6.11.6 ITrace::ReportFileInfo

```
HRESULT ReportFileInfo (
    [in] BSTR file_name )
```

Saves trace information into a specified HTML file.

### Parameters

<code>file_name</code>	String providing the full pathname to a file where the trace information report is stored
------------------------	---

### Return values

<code>ANALYZERCOMERROR_UNABLESAVE</code>	Unable to create the trace information report
--	---

### Remarks

Creates a new trace information file if the file specified in the *file\_name* parameter does not exist.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
Trace.ReportFileInfo CurrentDir & "Output\file_info.html"
```

C++:

```
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];

. . .

try
{
    pe_trace->ReportFileInfo( file_name );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

### 6.11.7 ITrace::ReportErrorSummary

```
HRESULT ReportErrorSummary (  
    [in] BSTR file_name )
```

Saves trace error summary information into the specified text file.

#### Parameters

<code>file_name</code>	String providing the full pathname to a file where the error summary report is stored.
------------------------	--

#### Return values

<code>ANALYZERCOMERROR_UNABLESAVE</code>	Unable to create trace information report
--	---

#### Remarks

This method doesn't work on Multisegment traces.

Creates a new error summary file if the file specified in the *file\_name* parameter does not exist. Stores error summary in the specified file.

Here is an example of data stored using this method call:

Error report for ErrorFinding\_loop.pex recording file.

```
|_____|
Bad ECRCs on channel Upstream (0):
|_____|
Bad ECRCs on channel Downstream (0):
|_____|
Bad LCRCs on channel Upstream (12):
|_____|
Bad LCRCs on channel Downstream (0):
|_____|
Bad Packet length on channel Upstream (0):
|_____|
Bad Packet length on channel Downstream (0):
|_____|
Alignment Error on channel Upstream (0):
|_____|
Alignment Error on channel Downstream (0):
|_____|
Invalid 10b Code on channel Upstream (11):
|_____|
Invalid 10b Code on channel Downstream (0):
|_____|
Running Disparity Error on channel Upstream (0):
|_____|
Running Disparity Error on channel Downstream (0):
|_____|
End of Bad Packet on channel Upstream (0):
|_____|
End of Bad Packet on channel Downstream (0):
|_____|
Delimiter Error on channel Upstream (12):
|_____|
Delimiter Error on channel Downstream (0):
|_____|
TS Data Error on channel Upstream (0):
|_____|
TS Data Error on channel Downstream (0):
|_____|
Ordered Set Format Error on channel Upstream (0):
|_____|
Ordered Set Format Error on channel Downstream (0):
|_____|
Idle Error on channel Upstream (0):
|_____|
Idle Error on channel Downstream (11):
|_____|
Skip Late on channel Upstream (0):
|_____|
Skip Late on channel Downstream (0):
|_____|
Skew Error on channel Upstream (0):
|_____|
Skew Error on channel Downstream (0):
|_____|
```

## Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
Trace.ReportErrorSummary CurrentDir & "Output\error_summary.txt"
```

C++:

```
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];

. . .

try
{
    pe_trace->ReportErrorSummary( file_name );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.11.8 ITrace::ReportTrafficSummary

```
HRESULT ReportTrafficSummary (
    [in] BSTR file_name )
```

Saves trace traffic summary information into the specified text file.

### Parameters

file_name	String providing the full pathname to a file where the traffic summary report is stored.
-----------	--

### Return values

ANALYZERCOMERROR_UNABLESAVE	Unable to create traffic summary report
-----------------------------	---

### Remarks

This method doesn't work on Multisegment traces.

Creates a new traffic summary file if the file specified in the *file\_name* parameter does not exist. Stores traffic summary in the specified file.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
Trace.ReportTrafficSummary CurrentDir & "Output\traffic_summary.txt"
```

C++:

```
IPETrace* pe_trace;
TCHAR file_name[_MAX_PATH];

. . .

try
{
    pe_trace->ReportTrafficSummary( file_name );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
```

## 6.11.9 ITrace::GetPacket

```
HRESULT GetPacket (
    [in] long packet_number,
    [in, out] VARIANT* packet,
    [out, retval] long* number_of_bytes )
```

Retrieves a raw packet representation in the *PACKETFORMAT\_BYTES* format (see *IPacket* interface for details, Page 85).

### Parameters

packet_number	Zero based number of packet to retrieve
packet	Raw packet representation
number_of_bytes	Number of bytes in the raw packet representation

### Return values

ANALYZERCOMERROR_INVALIDPACKETNUMBER	Specified packet number is invalid
--------------------------------------	------------------------------------

### Remarks

*packet* parameter has *VT\_ARRAY|VT\_VARIANT* actual automation type. Each element of this array has the *VT\_UI1* automation type.

## Example

VBScript:

```
<OBJECT
  ID = Analyzer
  CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>
<INPUT TYPE=TEXT NAME="TextPacketNumber">
<P ALIGN=LEFT ID=StatusText></P>

<SCRIPT LANGUAGE="VBScript">
<!--
Function DecToBin(Param, NeedLen)
  While Param > 0
    Param = Param/2
    If Param - Int(Param) > 0 Then
      Res = CStr(1) + Res
    Else
      Res = CStr(0) + Res
    End If
    Param = Int(Param)
  Wend
  DecToBin = Replace( Space(NeedLen - Len(Res)), " ", "0") & Res
End Function

Sub BtnGetPacket_OnClick
  On Error Resume Next
  Dim Packet
  NumberOfBytes = CurrentTrace.GetPacket (TextPacketNumber.value, Packet)
  If Err.Number <> 0 Then
    MsgBox "GetPacket:" & Err.Number & ":" & Err.Description
  Else
    For Each PacketByte In Packet
      PacketStr = PacketStr & DecToBin(PacketByte, 8) & " "
      NBytes = NBytes + 1
    Next
    PacketStr = Left( PacketStr, NumberOfBytes)
    StatusText.innerText = "Packet ( " & NumberOfBytes & " bytes ): " & PacketStr
  End If
End Sub
-->
</SCRIPT>
```

C++:

```

IPETrace* pe_trace;
LONG packet_number;

. . .

VARIANT packet;
VariantInit( &packet );
long number_of_bytes;
try
{
    number_of_bytes = pe_trace->GetPacket( packet_number, &packet );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

if ( packet.vt == ( VT_ARRAY | VT_VARIANT) )
{
    SAFEARRAY* packet_safearray = packet.parray;

    TCHAR packet_message[256];
    TCHAR elem[64];
    _stprintf( packet_message, _T("packet #%ld: "), packet_number );

    for ( long i=0; i<(long)packet_safearray->rgsabound[0].cElements; i++)
    {
        VARIANT var;
        HRESULT hr = SafeArrayGetElement(packet_safearray, &i, &var);
        if (FAILED(hr))
        {
            ::MessageBox( NULL, _T("Error accessing array"), _T("PETracer client"), MB_OK );
            return 1;
        }
        if ( var.vt != ( VT_UI1 ) )
        {
            ::MessageBox( NULL, _T("Array of bytes expected"), _T("PETracer client"), MB_OK );
            return 1;
        }

        _stprintf( elem, _T("%02X "), V_UI1(&var) );
        _tcscat( packet_message, elem );
    }
    _stprintf( elem, _T("%d bytes"), number_of_bytes );
    _tcscat( packet_message, elem );

    ::MessageBox( NULL, packet_message, _T("Raw packet bits"), MB_OK );
}
else
{
    ::MessageBox( NULL, _T("Invalid argument"), _T("PETracer client"), MB_OK );
}

```

### 6.11.10 ITrace::GetPacketsCount

```
HRESULT GetPacketsCount (
    [out, retval] long* number_of_packets )
```

Retrieves the total number of packets in the trace.

#### Parameters

number_of_packets	Total number of packets in the trace
-------------------	--------------------------------------

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
MsgBox Trace.GetPacketsCount & " packets recorded"
```

C++:

```
IPETrace* pe_trace;

. . .

long number_of_packets;
long trigg_packet_num;
try
{
    bstr_trace_name = pe_trace->GetName();
    number_of_packets = pe_trace->GetPacketsCount();
    trigg_packet_num = pe_trace->GetTriggerPacketNum();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

TCHAR str_trace_name[256];
_tcscpy( str_trace_name, (TCHAR*)( bstr_trace_name ) );
SysFreeString( bstr_trace_name );

TCHAR trace_info[256];
_stprintf( trace_info, _T("Trace:'%s', total packets:%ld, trigger packet:%ld"),
    str_trace_name, number_of_packets, trigg_packet_num );

::SetWindowText( m_hwndStatus, trace_info );
```

### 6.11.11 ITrace::GetTriggerPacketNum

```
HRESULT GetTriggerPacketNum (
    [out, retval] long* packet_number )
```

Retrieves the trigger packet number.

#### Parameters

packet\_number      Zero based number of the packet where the trigger occurred

#### Return values

#### Remarks

#### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
TriggerPacket = Trace.GetTriggerPacketNum
Trace.Save CurrentDir & "Output\trigger_portion.pex", CInt(ErrorPacket)-5,
    CInt(ErrorPacket)+5
```

C++:

See an example for *ITrace::GetPacketsCount*, Page 37.

## 6.11.12 ITrace::AnalyzerErrors

```
HRESULT AnalyzerErrors (
    [in] long error_type,
    [out, retval] IAnalyzerErrors** analyzer_errors )
```

Retrieves trace file errors. Returns an interface pointer to the *AnalyzerErrors* object.

### Parameters

error_type	Type of error collection you want to retrieve; the following values are valid:
	<pre> 0x00000001  Invalid Code 0x00000002  Running Disparity Error 0x00000003  Skip Late 0x00000004  Idle Data Error (not D0.0) 0x00000005  Unexpected K/D Code 0x00000006  Bad Packet Length 0x00000007  Alignment Error 0x00000008  Skew Error 0x00000009  Ordered Set Format Error 0x0000000A  Delimiter Error 0x0000000B  DLLP: Invalid Encoding 0x0000000C  DLLP: Bad CRC16 0x0000000D  DLLP: Reserved Field not 0 0x0000000E  DLLP: FC Initialization Error 0x0000000F  TLP: Invalid Encoding 0x00000010  TLP: Bad LCRC 0x00000011  TLP: Bad ECRC 0x00000012  TLP: Reserved Field not 0 0x00000013  TLP: Payload/Length Error 0x00000014  TLP: Length Error (not 1) 0x00000015  TLP: TC Error (not 0) 0x00000016  TLP: Attr Error (not 0) 0x00000017  TLP: Byte Enables Violation 0x00000018  Memory TLP: Address/Length Crosses 4K 0x00000019  Mem64 TLP: Used Incorrectly 0x0000001A  Cfg TLP: Register Error 0x0000001B  Msg TLP: Invalid Routing 0x0000001C  Invalid Packet 0x0000001D  FC: Invalid Advertisement 0x0000001E  FC: Insufficient Credits 0x0000001F  Gen3 TLP: Bad Len CRC/Parity 0x00000020  TLP: AT Error (not 0) .....0x00000021  Training Sequence Format Error .....0x00000022  Training Sequence Parity Error .....0x00000023  RRAP Command Aborted Error .....0x00000024  RRAP Packet Parity Error .....0x00000025  RRAP Packet Reserved Fields Not Nullified Error .....0x00000026  Msg TLP: Invalid Function Number field </pre>
analyzer_errors	Address of a pointer to the <i>AnalyzerErrors</i> object interface

### Return values

ANALYZERCOMERROR_INVALIDERROR	Invalid error type specified
-------------------------------	------------------------------

### Remarks

*AnalyzerErrors* object is created via this method call if the call was successful.

## Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
Set Errors = Trace.AnalyzerErrors( 8 ) ' Packet Length Error
```

C++:

```
IPETrace* pe_trace;
. . .

IAnalyzerErrors* trace_errors;
try
{
    trace_errors = pe_trace->AnalyzerErrors(error_type).Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}
. . .

analyser_errors->Release();
```

## 6.12 IPETrace interface

The *IPETrace* interface is a primary dual interface for the *PETrace* object.

This interface is derived from the *ITrace* interface.

The *IPETrace* interface implements all methods from the *ITrace* interface plus the following:  
*GetBusPacket*

### 6.12.1 IPETrace::GetBusPacket

```
HRESULT GetBusPacket (
    [in] long packet_number,
    [out, retval] IDispatch** packet )
```

Retrieves the interface for a packet within a trace.

#### Parameters

packet_number	Zero based number of packet to retrieve
packet	Address of a pointer to the <i>PEPacket</i> object interface

#### Return values

#### Remarks

*PEPacket* object is created via this method call if the call was successful.

#### Example

WSH:

C++:

```
IPETrace* pe_trace;

. . .

IDispatch* packet;
try
{
    packet = pe_trace->GetBusPacket( GetDlgItemInt(IDC_PACKET_NUMBER) ).Detach();
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
    return 1;
}

IPEPacket* custom_packet;
HRESULT hr = packet->QueryInterface( IID_IPEPacket, (void*)&custom_packet );
packet->Release();
```

## 6.13 IPETrace2 interface

The **IPETrace2** interface is an interface for the **PETrace** object, inheriting and extending **IPETrace** interface.

The *IPETrace2* interface implements all methods from the *IPETrace* interface plus the following:

*IsLevelDecodeFinished*  
*SetTag*  
*GetTag*

**IID** : 54DD9DFF-22DE-43D8-A124-04DF13ABE527

### 6.13.1 IPETrace2::IsLevelDecodeFinished

```
HRESULT IsLevelDecodeFinished(
    [in] long level_id,
    [out,retval] BOOL* decode_is_finished )
```

Retrieves progress status of decoding levels process. If level decoding is finished, returns true.

#### Parameters

`level_id` Desired level to retrieve decoding status like Transaction/Transfer  
`decode_is_finished` Pointer to Boolean value that specifies decoding status

#### Return values

#### Remarks

After opening traces, decoding process of upper levels starts in background processes or through a function call for higher levels. This decoding is time consuming process, however any other process on desired level like transactions need to check of decoding finish before starting process.

The level ID parameter could be one of these values:

DECODE_LEVEL_PACKET	= 0,	// PE_TRA_LEVEL_PACKETS	= 1
DECODE_LEVEL_LINKTRA	= 1,	//PE_TRA_LEVEL_LINKTRA	= 2
DECODE_LEVEL_SPLITTRA	= 2,	//PE_TRA_LEVEL_SPLITTRA	= 3
DECODE_LEVEL_NVMHCI	= 3,	//PE_TRA_LEVEL_NVMHCI	= 4
DECODE_LEVEL_PQI	= 4,	//PE_TRA_LEVEL_PQI	= 5
DECODE_LEVEL_AHCI	= 5,	//PE_TRA_LEVEL_AHCI	= 6
DECODE_LEVEL_ATA	= 6,	//PE_TRA_LEVEL_ATA	= 7
DECODE_LEVEL_SOP	= 7,	//PE_TRA_LEVEL_SOP	= 8
DECODE_LEVEL_SCSI	= 8,	//PE_TRA_LEVEL_SCSI	= 9
DECODE_LEVEL_NVC	= 9,	//PE_TRA_LEVEL_NVC	= 10
DECODE_LEVEL_RRAP	= 10,	//MPE_TRA_LEVEL_RRAP	= 11
DECODE_LEVEL_MCTP_MSG	= 11,	//PE_TRA_LEVEL_MCTP_MSG	= 12
DECODE_LEVEL_MCTP_CMD	= 12,	//PE_TRA_LEVEL_MCTP_CMD	

### 6.13.2 IPETrace2::SetTag

```
HRESULT SetTag( [in] int tag)
```

Specify a optional unique id for trace. This id could be used as identifier of trace in trace related events when user works with mutiple traces at same time. See `_ITraceEvent` interface for further detail.

#### Parameters

`tag` optional unique identifier of trace.

#### Return values

#### Remarks

### 6.13.3 IPETrace2::GetTag

```
HRESULT GetTag( [out, retval] int* tag)
```

Retrieves the unique identifer of trace which previously specified by `SetTag` method.

#### Parameters

`tag` optional unique identifier of trace.

#### Return values

#### Remarks

## 6.14 \_ITraceEvents interface

The **\_ITraceEvents** interface provides an event interface over trace files objects. This interface is used to through some events from trace analysis to client application. Be noted that this interface is not independent object and sink object in client code [who implements the events methods] should bind to existing trace object to notified by events from trace.

**IID** : 81C4C56A-AB57-4402-AD11-7D49F7CF450E

### 6.14.1 ITraceEvents::OnLevelDecodeFinished

```
HRESULT OnLevelDecodeFinished  
( [in] int levelId,  
  [in] int tag )
```

This event notifies client when decoding of any levels in applications like transaction/split/transfer... is finished. Client code may start some other process on this level data after getting finish decoding event.

#### Parameters

level_id	Specifies level that decoding process is finished
tag	Specified the unique of trace that previously set by SetTag

#### Return values

#### Remarks

For detail information about level IDs please refer to **IPETrace2::IsLevelDecodeFinished** section.

## 6.15 IPETrace3 Interface

The IPETrace3 interface is an additional interface for the PETrace object, inheriting and extending functionality exposed via the IPETrace2 interface.

The *IPETrace3* interface implements all methods from the *IPETrace2* interface plus the following:

*GetPacketEx*  
*GetPacketCountEx*  
*GetTriggerPacketNumEx*  
*GetBusPacketEx*

IID: 68227448-44B8-4144-A20F-FF9983CA6710

### 6.15.1 IPETrace3::GetPacketEx

```
HRESULT GetPacketEx
( [in] hyper packet_number,
  [in, out] VARIANT* packet,
  [out, retval] long* number_of_bits )
```

Retrieves a raw packet in the PACKETFORMAT\_BYTES format. It accepts 64 bits packet number. It is ITrace::GetPacket method extension.

#### Parameters

packet_number	Zero based number of packet to retrieve. It is a 64 bits integer.
packet	Raw packet representation
number_of_bytes	Number of bytes in the raw packet representation

#### Return values

ANALYZERCOMERROR\_INVALIDPACKETNUMBER Specified packet number is invalid

#### Remarks

Packet parameter has VT\_ARRAY | VT\_VARIANT actual automation type. Each elements of this array has the VT\_UI1 automation type.

### 6.15.2 IPETrace3::GetPacketsCountEx

```
HRESULT GetPacketsCountEx
([out, retval] hyper* number_of_packets )
```

Retrieves the total number of packets in the trace. Number of packets can be a 64 bits integer. It is ITrace::GetPacketsCount method extension.

#### Parameters

number_of_packets	Total number of packets in the trace
-------------------	--------------------------------------

#### Return values

#### Remarks

### 6.15.3 IPETrace3::GetTriggerPacketNumEx

```
HRESULT GetTriggerPacketNumEx
([out, retval] hyper* packet_number )
```

Retrieves the trigger packet number. The trigger packet number can be a 64 bits integer. It is *ITrace::GetTriggerPacketNum* method extension.

#### Parameters

`packets_number`      Zero based number of the packet where trigger occurred

#### Return values

#### Remarks

### 6.15.4 IPETrace3::GetBusPacketEx

```
HRESULT GetBusPacketEx
([in] hyper packet_number,
 [out, retval] IDispatch** packet)
```

Retrieves the interface for packet within a trace. It accepts 64-bit packet number. It is *IPETrace::GetBusPacket* method extension.

#### Parameters

`packet_number`      Zero based number of packet to retrieve. It is a 64 bits integer.  
`packet`              Address of a pointer to the PEPacket object interface

#### Return values

`ANALYZERCOMERROR_INVALIDPACKETNUMBER`    Incorrect packet number is passed as a parameter  
`ANALYZERCOMERROR_UNABLEOPENFILE`        Unable to open segment trace

#### Remarks

PEPacket object is created via this method if the call was successful.

## 6.16 IPEVerificationScript interface

The *IPEVerificationScript* interface is an interface for the *PETrace* object. It exposes the trace functionality for running verification scripts. This interface is not dual – which means that scripting languages cannot use it directly, though all of its methods described below are exposed to script languages through the primary automation interface of the *PETrace* object.

### Remarks

Verification scripts are scripts written in a special manner using the *CATC Script Language (CSL)*. These scripts can be “run” over a recorded trace to “verify” the trace for some verification conditions or to extract more advanced information from the trace. Such scripts utilize a special feature of the PCIe Protocol Analysis application, its *Verification Script Engine*.

Please refer to the *PCIe Protocol Analysis Manual*, the *PCIe Protocol Analysis Verification Script Engine Manual*, and the *PCIe Protocol Analysis File Based Decoding Manual* for more details.

### Attention:

The functions of this interface may be legally called either for regular traces or multi-segmented traces. The VSE opens segments of the multi-segmented trace during script execution when it is needed.

## 6.16.1 IPEVerificationScript::RunVerificationScript

```
HRESULT RunVerificationScript (
    [in] BSTR verification_script,
    [out, retval] VS_RESULT *result )
```

Runs a verification script over the recorded trace

### Parameters

verification_script	Name of the verification script to run																		
result	Address of a variable where to keep the result of verification; <i>VS_RESULT</i> is an enumeration type that can have 6 possible meanings:																		
	<table> <tr> <td>FEATURE_NOT_LICENSED</td> <td>( -3 )</td> <td>- feature is not licensed</td> </tr> <tr> <td>SCRIPT_RUNNING</td> <td>( -2 )</td> <td>- verification script is running</td> </tr> <tr> <td>SCRIPT_NOT_FOUND</td> <td>( -1 )</td> <td>- verification script with the specified name was not found</td> </tr> <tr> <td>FAILED</td> <td>( 0 )</td> <td>- verification failed</td> </tr> <tr> <td>PASSED</td> <td>( 1 )</td> <td>- verification passed</td> </tr> <tr> <td>DONE</td> <td>( 2 )</td> <td>- verification is done, don't care about result</td> </tr> </table>	FEATURE_NOT_LICENSED	( -3 )	- feature is not licensed	SCRIPT_RUNNING	( -2 )	- verification script is running	SCRIPT_NOT_FOUND	( -1 )	- verification script with the specified name was not found	FAILED	( 0 )	- verification failed	PASSED	( 1 )	- verification passed	DONE	( 2 )	- verification is done, don't care about result
FEATURE_NOT_LICENSED	( -3 )	- feature is not licensed																	
SCRIPT_RUNNING	( -2 )	- verification script is running																	
SCRIPT_NOT_FOUND	( -1 )	- verification script with the specified name was not found																	
FAILED	( 0 )	- verification failed																	
PASSED	( 1 )	- verification passed																	
DONE	( 2 )	- verification is done, don't care about result																	

### Return values

S\_OK If the verification script executed successfully.

### Remarks

The name of the verification script is the name of the verification script file (\*.pevs). If only the name of the script, without file extension, is specified, PCIe Protocol Analysis server is going to search for the named script among the scripts loaded from the \Scripts\VFScripts folder under PCIe Protocol Analysis installation folder. If the full path to the script is specified, then the server is going to attempt loading the script from the specified path prior to running it.

### Example

For a verification script file named "test.pevs", the test name would be "test". Please refer to the *PCIe Protocol Analysis Verification Script Engine Manual* for more details.

**Example**

C++:

```

// In this example we use wrapper functions provided by #import directive
//
IPETrace* trace;
. . .

IPEVerificationScript* vscrip = NULL;

if ( SUCCEEDED ( trace->QueryInterface( IID_IPEVerificationScript, (void**)&vscrip ) )
{
    try
    {
        VS_RESULT result = vscrip ->RunVerificationScript("Test1");
        if( result == PASSED )
        {
            ::MessageBox( NULL, "Test verification 1 is passed !!!", "PETracer client",
                MB_OK );
        }
    }
    catch ( _com_error& er)
    {
        if (er.Description().length() > 0)
            ::MessageBox( NULL, er.Description(), "PETracer client", MB_OK );
        else
            ::MessageBox( NULL, er.ErrorMessage(), "PETracer client", MB_OK );
        return 1;
    }
}
else
{
    ::MessageBox( NULL, "Unable to get IPEVerificationScript interface !!!",
        _T("PETracer client"), MB_OK );
    return 1 ;
}
. . .

```

WSH:

```

Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set Trace    = Analyzer.OpenFile( "C:\Some trace files\some_trace.pex" )

Dim Result
Result = Trace.RunVerificationScript( "Test1" )

If Result = 1 Then
    MsgBox "PASSED"
Else
    MsgBox "FAILED"
End If

MsgBox( "Done" )

```

## 6.16.2 IPEVerificationScript::GetVScriptEngine

```
HRESULT GetVScriptEngine(  
    [in] BSTR script_name,  
    [out, retval] IVScriptEngine** vs_engine )
```

Retrieves the verification script engine object

### Parameters

script_name	Name of the verification script to initialize the verification script engine
vs_engine	Address of a pointer to the <i>PEVScriptEngine</i> object interface

### Return values

S_OK	If the verification script engine object was successfully retrieved.
------	--

### Remarks

The name of the verification script is the name of the verification script file (\*.pevs). See remark to *IPEVerificationScript::RunVerificationScript* function for details, Page 43.

## Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vsript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vsript )
assert( pe_vsript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vsript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

VS_RESULT result = pe_vsengine ->RunVScript();
if( result == PASSED )
{
    ::MessageBox( NULL, "Test verification 1 is passed !!!", "PETracer client", MB_OK );
}

. . .
```

WSH:

```
Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set Trace    = Analyzer.OpenFile( "C:\Some trace files\some_trace.pex" )

Dim Result

Set VSEngine = Trace.GetVScriptEngine( "Test1" )
Result = VSEngine.RunVScript

If Result = 1 Then
    MsgBox "PASSED"
Else
    MsgBox "FAILED"
End If

MsgBox( "Done" )
```

## 6.17 IPETraceForScript2 interface

The IPETraceForScript2 interface is an additional interface for the IPETraceForScript object. It inherits and extends some trace-related functionality exposed via the IPETraceForScript interface.

IID: 1FFC22F5-705F-46BB-8B24-4865109B9136

### 6.17.1 IPETraceForScript2::ImportXMLConfig

```
HRESULT ImportXMLConfig(
    [in] BSTR xml,
    [in] EXmlConfigType xml_config_type,
    [in, defaultvalue(TRUE)] BOOL do_redecode );
```

Imports xml-config file from string

#### Parameters

xml	XML-to import
xml_config_type	Type of xml-config; EXmlConfigType is an enumeration type that can have 3 possible meanings: <ul style="list-style-type: none"> <li>USER_CONFIG ( 0 ) - User defined config</li> <li>INIT_CONFIG ( 1 ) - Initial config</li> <li>DECODING_CONFIG ( 2 ) - Decoding config</li> </ul>
do_redecode	Specifies if redecoding should be done just after importing. Default value is TRUE

## 6.17.2 IPETraceForScript2:: ImportXMLConfigFromFile

```
HRESULT ImportXMLConfigFromFile (  
    [in] BSTR xml_file_path,  
    [in] EXmlConfigType xml_config_type,  
    [in, defaultvalue(TRUE)] BOOL do_redecode );
```

Imports xml-config file from file

### Parameters

xml_file_path	Path to XML-file to import
xml_config_type	Type of xml-config. EXmlConfigType is an enumeration type that can have 3 possible meanings: USER_CONFIG                   ( 0 ) - User defined config INIT_CONFIG                   ( 1 ) - Initial config DECODING_CONFIG               ( 2 ) - Decoding config
do_redecode	Specifies if redecoding should be done just after importing. Default value is TRUE

### 6.17.3 IPETraceForScript2:: ExportXMLConfig

```
HRESULT ExportXMLConfig(  
    [in] EXmlConfigType xml_config_type,  
    [out, retval] BSTR* xml );
```

Exports xml-config file to string

#### Parameters

xml_config_type	Type of xml-config; EXmlConfigType is an enumeration type that can have 3 possible meanings:
	USER_CONFIG ( 0 ) - User defined config
	INIT_CONFIG ( 1 ) - Initial config
	DECODING_CONFIG ( 2 ) - Decoding config
xml	Returns xml-config as sting

### 6.17.4 IPETraceForScript2:: ExportXMLConfigToFile

```
HRESULT ExportXMLConfigToFile(  
    [in] BSTR xml_file_path,  
    [in] EXmlConfigType xml_config_type );
```

Exports xml-config file to file

#### Parameters

<code>xml_file_path</code>	Path to XML-file to export
<code>xml_config_type</code>	Type of xml-config; <code>EXmlConfigType</code> is an enumeration type that can have 3 possible meanings: <code>USER_CONFIG</code> ( 0 ) - User defined config <code>INIT_CONFIG</code> ( 1 ) - Initial config <code>DECODING_CONFIG</code> ( 2 ) - Decoding config

## 6.17.5 IPETraceForScript2:: Redecode

```
HRESULT Redecode(
    [in] EPETraceDecodeLevels level_to_redecode);
```

Decodes undecoded and redecodes decoded levels up to level\_to\_redecode.

### Parameters

level_to_redecode	Type of xml-config; EPETraceDecodeLevels is an enumeration type that can have 11 possible meanings:
	DECODE_LEVEL_PACKET ( 0 ) - Decoding Level Packet
	DECODE_LEVEL_LINKTRA ( 1 ) - "Decoding Level Link Transaction
	DECODE_LEVEL_SPLITTRA ( 2 ) - Decoding Level Split Transaction
	DECODE_LEVEL_NVMHCI ( 3 ) - Decoding Level NVM Transaction
	DECODE_LEVEL_PQI ( 4 ) - Decoding Level PQI
	DECODE_LEVEL_AHCI ( 5 ) - Decoding Level AHCI
	DECODE_LEVEL_ATA ( 6 ) - Decoding Level ATA
	DECODE_LEVEL_SOP ( 7 ) - Decoding Level SOP
	DECODE_LEVEL_SCSI ( 8 ) - Decoding Level SCSI
	DECODE_LEVEL_NVMC ( 9 ) - Decoding Level NVM Command
	DECODE_LEVEL_RRAP ( 10 ) - Decoding Level RRAP
	DECODE_LEVEL_MCTP_MSG ( 11 ) - Decoding Level MCTP MSG
	DECODE_LEVEL_MCTP_CMD ( 12 ) - Decoding Level MCTP CMD

## 6.18 IPETraceForScript3 interface

The IPETraceForScript3 interface is an additional interface for the IPETraceForScript2 object. It inherits and extends some trace-related functionality exposed via the IPETraceForScript interface.

IID: AA8FBCC1-065E-46AB-975E-989714293323

### 6.18.1 IPETraceForScript3:: ExportToText2

```
HRESULT ExportToText2(
    [in] BSTR file_name,
    [in, defaultvalue(-1)] hyper index_from,
    [in, defaultvalue(-1)] hyper index_to,
    [in, defaultvalue(DECODE_LEVEL_PACKET)] EPETraceDecodeLevels level );
```

Exports selected level to text file in selected range.

#### Parameters

file_name	Export file path
index_from	Transaction export to export from
index_to	Transaction export to export to
level	Type of xml-config; EPETraceDecodeLevels is an enumeration type that can have 11 possible meanings: <ul style="list-style-type: none"> <li>DECODE_LEVEL_PACKET ( 0 ) - Decoding Level Packet</li> <li>DECODE_LEVEL_LINKTRA ( 1 ) - "Decoding Level Link Transaction</li> <li>DECODE_LEVEL_SPLITTRA ( 2 ) - Decoding Level Split Transaction</li> <li>DECODE_LEVEL_NVMHCI ( 3 ) - Decoding Level NVM Transaction</li> <li>DECODE_LEVEL_PQI ( 4 ) - Decoding Level PQI</li> <li>DECODE_LEVEL_AHCI ( 5 ) - Decoding Level AHCI</li> <li>DECODE_LEVEL_ATA ( 6 ) - Decoding Level ATA</li> <li>DECODE_LEVEL_SOP ( 7 ) - Decoding Level SOP</li> <li>DECODE_LEVEL_SCSI ( 8 ) - Decoding Level SCSI</li> <li>DECODE_LEVEL_NVMC ( 9 ) - Decoding Level NVM Command</li> <li>DECODE_LEVEL_RRAP ( 10 ) - Decoding Level RRAP</li> <li>DECODE_LEVEL_MCTP_MSG ( 11 ) - Decoding Level MCTP MSG</li> <li>DECODE_LEVEL_MCTP_CMD ( 12 ) - Decoding Level MCTP CMD</li> </ul> Default value is DECODE_LEVEL_PACKET.

## 6.19 IPETraceForScript4 interface

The IPETraceForScript4 interface is an additional interface for the IPETraceForScript3 object. It inherits and extends some trace-related functionality exposed via the IPETraceForScript interface.

IID: 28811AAD-66E8-451C-9D26-A78496DBE71F

### 6.19.1 IPETraceForScript4:: GetTrafficSummaryAsXML

```
HRESULT GetTrafficSummaryAsXML(  
    [in] BSTR xml_file_path );
```

Exports Traffic Summary to the XML-file

#### Parameters

xml_file_path	Export XML file path
---------------	----------------------

## 6.20 IPETraceForScript5 interface

The IPETraceForScript5 interface is an additional interface for IPETraceForScript4 object. It inherits and extends functionality for exporting link tracker data for CSV.

IID: 6824470C-3C1F-43D6-AFE7-237F36687F08

### 6.20.1 IPETraceForScript5::ExportLinkTrackerToCsv

```
HRESULT ExportLinkTrackerToCsv (
    [in] BSTR export_filename,
    [in] int timestamp_format,
    [in] int collapse_mode,
    [in] int show_type );
```

Exports Link Tracker data to the CSV-file with specified options. Incorrect options or other export errors produces “Invalid argument” error with text description.

See specification for option details (Chapter 8. Report and Tools, 8.5 Link Tracker, Link Tracker Button).

#### Parameters

<code>export_filename</code>	CSV file path
<code>timestamp_format</code>	Timestamp format: <ul style="list-style-type: none"> <li>• 1 : seconds time format.</li> <li>• 2 : clocks time format.</li> </ul>
<code>collapse_mode</code>	Collapse mode format: <ul style="list-style-type: none"> <li>• 0 : continuous time scale. Attention: export could take many time.</li> <li>• 1 : collapse idle time.</li> <li>• 2 : collapse idle time plus.</li> </ul>
<code>show_type</code>	Show format: <ul style="list-style-type: none"> <li>• 0 : .bytes.</li> <li>• 1 : 10-bit codes.</li> <li>• 2 : symbols.</li> <li>• 3 : packet fields.</li> </ul>

## 6.21 IPETraceForScript6 interface

The IPETraceForScript6 interface is an additional interface for the IPETraceForScript5 object. It inherits and extends some trace-related functionality exposed via the IPETraceForScript interface.

IID: C3D83CB4-3F33-11E5-A151-FEFF819CDC9F

### 6.21.1 IPETraceForScript5:: SetOldExpFormat

```
HRESULT SetOldExpFormat (  
    [in] int is_old_export_format);
```

Sets export format: old or new one.

Changes in export format since 7.38 version:

- 1) No spaces in time stamp format
- 2) 'Data' field units changed from 'Dword' to 'Byte'
- 3) Some NVMe fields (QID, CID etc.) now represented in DEC instead of HEX

#### Parameters

old\_export\_format      TRUE for using legacy format, FALSE otherwise.

## 7 PERecOptions Object

The *PERecOptions* object represents the options for the Summit Analyzer hardware and is used to specify the recording parameters.

The *PERecOptions* object allows user to:

- Load/save the recording options from/to the file
- Set up recording mode and recording buffer size
- Set up custom recording parameters such as link width, descrambling mode, deskew, etc.

The *PERecOptions* object can be created by using the *IAnalyzer::GetRecordingOptions* method (see Page 15)

The *PERecOptions* object supports the following interfaces:

Interfaces	Description
<i>IRecOptions</i>	Allows you to load/save recording options from/to the file, reset recording options, set up recording mode, recording buffer size, trigger position, and the trace file name
<i>IPERecOptions</i>	Identical to <i>IRecOptions</i> interface
<i>IPERecOptions2</i>	Extends the <i>IPERecOptions</i> interface. Adds a set up for link width, spec mode, external reference clock, descrambling algorithm, skew, lane reversal, and polarity inversion

The *IPERecOptions2* interface is a primary interface for *PERecOptions* object.

## 7.1 IRecOptions interface

The *IRecOptions* interface is a dual interface for *PERecOptions* object.

*IRecOptions* implements the following methods:

```
Load
Save
SetRecMode
SetBufferSize
SetPostTriggerPercentage
SetTriggerBeep
SetSaveExternalSignals
SetTraceFileName
Reset
```

**Note:** All methods of the *IRecOptions* interface are also available in the *IPERecOptions* (see Page 57) and the *IPERecOptions2* (see Page 57) interfaces.

### 7.1.1 IRecOptions::Load

```
HRESULT Load (
    [in] BSTR ro_file_name )
```

Loads recording options from the specified file. The load options are:

- Loads the whole Recording Options (.rec)
- Loads only the Recording Rules portions of the recording options (.rr)
- Loads only the Probe Settings portions of the recording options (.ps). This option only applies to the Summit T3-16, T3-8, T28 and T24 Protocol Analyzers.

#### Parameters

`ro_file_name`                      String that provides the full pathname to the recording options file

#### Return values

`ANALYZERCOMERROR_UNABLEOPENFILE`                      Unable to open file

#### Remarks

#### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.Load( CurrentDir & "Input\rec_options.rec" )
```

C++:

## 7.1.2 IRecOptions::Save

```
HRESULT Save (
    [in] BSTR ro_file_name )
```

Saves recording options into the specified file. The Save options are:

- Saves the whole Recording Options (.rec)
- Saves only the Recording Rules portions of the recording options (.rr)
- Saves only the Probe Settings portions of the recording options (.ps). This option only applies to the Summit T3-16, T3-8, T28 and T24 Protocol Analyzers.

### Parameters

ro file name                      String that provides the full pathname to the recording options file

### Return values

ANALYZERCOMERROR\_UNABLEOPENFILE      Unable to open file

### Remarks

If the specified file does not exist, it is created; if it exists, it is overwritten.

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
' do the changes of recording options here
RecOptions.Save( CurrentDir & "Input\rec_options.rec" )
```

C++:

### 7.1.3 IRecOptions::SetRecMode

```
HRESULT SetRecMode (
    [in] ERecModes rec_mode )
```

Sets the recording mode.

#### Parameters

rec_mode	Enumerated value providing the mode to set; <i>ERecModes</i> enumerator has the following values:
	RMODE_SNAPSHOT ( 0 ) – snapshot recording mode
	RMODE_MANUAL ( 1 ) – manual trigger
	RMODE_USE_TRG ( 2 ) – event trigger

#### Return values

E_INVALIDARG	Invalid recording mode was specified
--------------	--------------------------------------

#### Remarks

The default setting of recording options is a “snapshot” recording mode.

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetRecMode 2 ' Event trigger
```

C++:

## 7.1.4 IRecOptions::SetBufferSize

```
HRESULT SetBufferSize (
    [in] long buffer_size )
```

Sets the size of buffer to record.

### Parameters

buffer_size	Size of the recording buffer in bytes
-------------	---------------------------------------

### Return values

E_INVALIDARG	Invalid buffer size was specified
--------------	-----------------------------------

### Remarks

The default setting will be used from the buffer size specified in the recording options file.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetBufferSize 2*1024*1024 ' 2Mb
```

C++:

### 7.1.5 IRecOptions::SetBufferSizeEx

```
HRESULT SetBufferSizeEx (  
    [in] unsigned long buffer_size )
```

Sets the size of buffer to record.

#### Parameters

buffer_size	Size of the recording buffer in mega bytes
-------------	--

#### Return values

E_INVALIDARG	Invalid buffer size was specified
--------------	-----------------------------------

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set RecOptions = Analyzer.GetRecordingOptions( )  
RecOptions.SetBufferSizeEx 2 ' 2Mb
```

## 7.1.6 IRecOptions::SetPostTriggerPercentage

```
HRESULT SetPostTriggerPercentage (
    [in] short posttrigger_percentage )
```

Sets the post trigger buffer size.

### Parameters

**posttrigger\_percentage**      Size of the post trigger buffer in percent of the whole recording buffer (see *IRecOptions::SetBufferSize*, Page 51)

### Return values

**E\_INVALIDARG**                      Invalid percentage was specified

### Remarks

This method call has no effect if recording mode was set to `RMODE_SNAPSHOT` (see *IRecOptions::SetRecMode*, Page 50). The default setting is 50%.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetPostTriggerPercentage 60              ' 60%
```

C++:

## 7.1.7 IRecOptions::SetTriggerBeep

```
HRESULT SetTriggerBeep (
    [in] BOOL beep )
```

Sets a flag to make a sound when a trigger occurs.

### Parameters

beep	TRUE – Beep when a trigger occurs, FALSE – Do not beep when a trigger occurs.
B	

### Return values

### Remarks

The default state of the beeper is FALSE.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetTriggerBeep TRUE
```

C++:

## 7.1.8 IRecOptions::SetSaveExternalSignals

```
HRESULT SetSaveExternalSignals (
    [in] BOOL save )
```

Sets a flag to save external signals.

### Parameters

save	TRUE – save external signals, FALSE – do not save external signals
------	---

### Return values

### Remarks

By default, external signals are not saved.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetSaveExternalSignals TRUE
```

C++:

## 7.1.9 IRecOptions::SetTraceFileName

```
HRESULT SetTraceFileName (
    [in] BSTR file_name )
```

Sets the file path to where the trace is stored after recording.

### Parameters

file_name	String that provides the full file pathname to where the recording is stored
-----------	--

### Return values

### Remarks

If the specified file does not exist, it is created; if it exists, it is overwritten.

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions( )
' do the changes of recording options here
RecOptions.Save( CurrentDir & "Input\trace.pex" )
```

C++:

## 7.1.10 IRecOptions::Reset

```
HRESULT Reset ( )
```

Resets the recording options to the initial state.

### Parameters

### Return values

### Remarks

For default values of recording options, see the remarks sections of all *IRecOptions*, *IPERecOptions*, and *IPERecOptions2* methods.

### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetRecMode 2 ' Event trigger
RecOptions.SetBufferSize 1024*1024 ' 1Mb
RecOptions.SetPostTriggerPercentage 60 ' 60%
. . .
RecOptions.Reset
```

C++:

## 7.2 IPERecOptions interface

This interface is identical to the *IRecOptions* interface (see Page 48).

## 7.3 IPERecOptions2 interface

The *IPERecOptions2* interface is a primary dual interface for the *PERecOptions* object.

This interface is derived from the *IPERecOptions* interface.

The *IPERecOptions2* interface implements all methods from the *IPERecOptions* interface, plus the following:

- SetTargetAnalyzer*
- SetLinkWidth*
- SetBase10Spec*
- SetExternalRefClock*
- SetDisableDescrambling*
- SetDisableDeskew*
- SetAutoConfigPolarity*
- SetInhibit*
- SetReverseLanes*
- SetInvertPolarity*

### 7.3.1 IPERecOptions2::SetTargetAnalyzer

```
HRESULT SetTargetAnalyzer(
    [in] ETargetAnalyzer target_analyzer )
```

Sets the hardware configuration for the recording options.

#### Parameters

target_analyzer	Enumerated value that provides the platform to set; <i>ETargetAnalyzer</i> has the following values:
	TARGETANALYZER_MP (7) - Summit T3-16
	TARGETANALYZER_MPX8 (8) - Summit T3-8
	TARGETANALYZER_MPSP (9) - Summit T28
	TARGETANALYZER_MPx8_2 (10) - Summit T3-8 (2 boxes)
	TARGETANALYZER_MPHR (11) - Summit T24
	TARGETANALYZER_T34 (12) - Summit T34
	TARGETANALYZER_T34_2 (13) - Summit T34 (2 boxes)
	TARGETANALYZER_MPHY_X34 (14) - Eclipse X34
	TARGETANALYZER_MPHY_X34_2 (15) - Eclipse X34 (2 boxes)
	TARGETANALYZER_Z4 (16) - Summit Z416
	TARGETANALYZER_T416 (17) - Summit T416
	TARGETANALYZER_T48 (18) - Summit T48
	TARGETANALYZER_Z4PTC (19) - Summit Z4-1PTC
	TARGETANALYZER_M5x (20) - Summit M5x

### 7.3.2 IPERecOptions2::SetLinkWidth

```
HRESULT SetLinkWidth (
    [in] int link_width )
```

Sets link width in recording options.

#### Parameters

Parameter	Values	Comments	Applicable Analyzer
link_width	1	Fixed x1	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x
	2	Fixed x2	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x
	4	Fixed x4	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x
	8	Fixed x8	T28, T34 (2), T3-8, T3-16, Z416, T416, T48, M5x
	16	Fixed x16	T3-8 (2), T3-16, Z416, T416, M5x
	20	Auto (x2 Maximum)	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x
	40	Auto (x4 Maximum)	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x
	80	Auto (x8 Maximum)	T28, T34 (2), T3-8, T3-16, M5x
	160	Auto (x16 Maximum)	T3-8 (2), T3-16, Z416, T416, M5x
	0	Auto (for backward compatibility. Maximum link width will be that of the attached Summit Analyzer)	T24, T28, T34, T3-8, T3-16, Z416, T416, T48, M5x

### 7.3.3 IPERecOptions2::SetBase10Spec

```
HRESULT SetBase10Spec (  
    [in] BOOL base_10_spec )
```

Sets PCI Express Base Specification 1.0 compatibility mode.

#### Parameters

base_10_spec	When TRUE, the Summit Analyzer hardware uses Base Spec 1.0 compatibility mode
--------------	---

#### Remarks

Not implemented for newer generation analyzers.

### 7.3.4 IPERecOptions2::SetExternalRefClock

```
HRESULT SetExternalRefClock(  
    [in] BOOL ext_ref_clock )
```

Specifies whether to use the external or internal reference clock

#### Parameters

ext_ref_clock	When TRUE, the external reference clock is used
---------------	---

### 7.3.5 IPERecOptions2::SetDisableDescrambling

```
HRESULT SetDisableDescrambling (  
    [in] BOOL disable_descrambling )
```

Disables/enables descrambling of incoming traffic.

#### Parameters

disable_descrambling	When TRUE, the descrambling is disabled
----------------------	---

#### Remarks

### 7.3.6 IPERecOptions2::SetDisableDeskew

```
HRESULT SetDisableDeskew(  
    [in] BOOL disable_deskew )
```

Disables/enables deskew of incoming traffic.

**Parameters:**

disable\_deskew      When TRUE, the deskew is disabled

**Remarks**

### 7.3.7 IPERecOptions2::SetAutoConfigPolarity

```
HRESULT SetAutoConfigPolarity(  
    [in] BOOL auto_config )
```

Enables/disables automatic polarity detection.

**Parameters:**

auto\_config                      When TRUE, lane polarity is detected automatically for all lanes.

**Remarks**

### 7.3.8 IPERecOptions2::SetInhibit

```
HRESULT SetInhibit(  
    [in] EDirection direction,  
    [in] BOOL inhibit )
```

Inhibits one of the traffic directions.

#### Parameters:

direction	Enumerated value that provides traffic direction to inhibit; <i>EDirection</i> has the following values: DIRECTION_UPSTREAM (0) - upstream traffic DIRECTION_DOWNSTREAM (1) - downstream traffic
inhibit	Specifies whether to inhibit traffic specified in the <i>direction</i> parameter

#### Remarks

### 7.3.9 IPERecOptions2::SetReverseLanes

```
HRESULT SetReverseLanes(  
    [in] EDirection direction,  
    [in] BOOL reverse )
```

Allows lane reversal on the specified traffic direction.

#### Parameters:

direction	Enumerated value that provides traffic direction for lane reversal; <i>EDirection</i> has the following values: DIRECTION_UPSTREAM (0) - upstream traffic DIRECTION_DOWNSTREAM (1) - downstream traffic
reverse	Specifies whether to reverse the lanes in the direction selected.

#### Remarks

### 7.3.10 IPERecOptions2::SetInvertPolarity

```
HRESULT SetInvertPolarity (  
    [in] EDirection direction,  
    [in] int lane,  
    [in] BOOL invert )
```

Allows polarity inversion of the specified lane and specified traffic direction.

#### Parameters:

direction	Enumerated value that provides traffic direction for polarity inversion; <i>EDirection</i> has the following values: DIRECTION_UPSTREAM (0) - upstream traffic DIRECTION_DOWNSTREAM (1) - downstream traffic
lane	Specifies the lane for polarity inversion
invert	Sets polarity inversion on the specified lane and specified link direction

#### Remarks

This call fails if automatic polarity detection is enabled (see *IPERecOptions2::SetAutoConfigPolarity*, Page 64).

## 7.4 IPERecOptions3 interface

The *IPERecOptions3* interface is a primary dual interface for the *PERecOptions* object.

This interface is derived from the *IPERecOptions* interface.

The *IPERecOptions3* interface implements all methods from the *IPERecOptions* interface, plus the following:

- SetLinkSpeed*
- GetSimpleTrigger*
- GetSimpleFilter*
- RestoreDefaultFactorySettings*

### 7.4.1 IPERecOptions3::SetLinkSpeed

```
HRESULT SetLinkSpeed ( [in] ERecOptSpeed e_link_speed )
```

Sets the link speed for the recording options.

#### Parameters:

link_speed	- enum that can have following values:
	SPD_AUTO_DETECT ( 0 )
	SPD_G1 ( 1 )
	SPD_G2 ( 2 )
	SPD_G3 ( 3 )

## 7.4.2 IPERecOptions3::GetSimpleTrigger

```
HRESULT GetSimpleTrigger ( [in] ETriggerType e_trig_type, [in] BOOL b_force, [out,
    retval] IDispatch** ppISimpleTrigger)
```

Returns simple trigger interface of the given trigger type.

### Parameters:

e_trig_type	enum that can have following values:	
	TRIGGER_EVENTTYPE_ERROR_ANY	( 1 )
	TRIGGER_EVENTTYPE_LINKSTATE	( 2 )
	TRIGGER_EVENTTYPE_FTS	( 3 )
	TRIGGER_EVENTTYPE_TLP_CONFIG_RD	( 4 )
	TRIGGER_EVENTTYPE_TLP_CONFIG_WR	( 5 )
	TRIGGER_EVENTTYPE_TLP_IO_RD	( 6 )
	TRIGGER_EVENTTYPE_TLP_IO_WR	( 7 )
	TRIGGER_EVENTTYPE_TLP_MEM_RD	( 8 )
	TRIGGER_EVENTTYPE_TLP_MEM_WR	( 9 )
	TRIGGER_EVENTTYPE_TLP_COMPLETION	( 10 )
	TRIGGER_EVENTTYPE_DLLP_INIT_FC2	( 11 )
	TRIGGER_EVENTTYPE_DLLP_ACK	( 12 )
	TRIGGER_EVENTTYPE_TS1	( 101 )
	TRIGGER_EVENTTYPE_TS2	( 102 )
	TRIGGER_EVENTTYPE_TLP_ANY	( 103 )
	TRIGGER_EVENTTYPE_TLP_MESSAGE	( 104 )
	TRIGGER_EVENTTYPE_DLLP_INIT_FC1	( 105 )
	TRIGGER_EVENTTYPE_DLLP_NAK	( 106 )
	TRIGGER_EVENTTYPE_PERRET_DLLP_POWER_MANAGEMENT	( 107 )
	TRIGGER_EVENTTYPE_TLP_CONFIG	( 200 )
	TRIGGER_EVENTTYPE_TLP_IO	( 201 )
	TRIGGER_EVENTTYPE_TLP_MEM	( 202 )
	TRIGGER_EVENTTYPE_LS_ENTER_EI	( 203 )
	TRIGGER_EVENTTYPE_LS_EXIT_EI	( 204 )
	TRIGGER_EVENTTYPE_LS_SPEEDCHANGE_G1	( 205 )
	TRIGGER_EVENTTYPE_LS_SPEEDCHANGE_G2	( 206 )
	TRIGGER_EVENTTYPE_LS_SPEEDCHANGE_G3	( 207 )
	TRIGGER_EVENTTYPE_ERROR_SYMBOL	( 209 )
	TRIGGER_EVENTTYPE_ERROR_DISPARITY	( 208 )
	TRIGGER_EVENTTYPE_ERROR_BLOCK_ALIGN	( 210 )
	TRIGGER_EVENTTYPE_ERROR_TOKEN	( 211 )
	TRIGGER_EVENTTYPE_ERROR_IDLE	( 212 )
	TRIGGER_EVENTTYPE_LS_CLKREQ_F	( 213 )
	TRIGGER_EVENTTYPE_LS_CLKREQ_R	( 214 )
	TRIGGER_EVENTTYPE_LS_WAKE_F	( 215 )
	TRIGGER_EVENTTYPE_LS_WAKE_R	( 216 )
	TRIGGER_EVENTTYPE_LS_LWC_X1	( 217 )
	TRIGGER_EVENTTYPE_LS_LWC_X2	( 218 )
	TRIGGER_EVENTTYPE_LS_LWC_X4	( 219 )
	TRIGGER_EVENTTYPE_LS_LWC_X8	( 220 )
	TRIGGER_EVENTTYPE_LS_LWC_X16	( 221 )

TRIGGER_EVENTTYPE_LS_PERST_F	( 222 )
TRIGGER_EVENTTYPE_LS_PERST_R	( 223 )

b\_force - boolean that can be 1 or 0 (1 is for enforcing even if the advanced trigger already exists. In this case the trigger will be converted to the simple type);

### Return values

SimpleTrigger object

### Remarks

### Example

```
Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set recOpt = Analyzer.GetRecordingOptions()
recOpt.SetRecMode(2)
ENFORCE = 1
TRIGGER_EVENTTYPE_TS2 = 102
Set simpleTrigger = recOpt.GetSimpleTrigger(TRIGGER_EVENTTYPE_TS2, ENFORCE)
```

See [7.5 SimpleTrigger](#) for more information.

### 7.4.3 IPERecOptions3::GetSimpleFilter

```
HRESULT GetSimpleFilter ( [in] EFilterType e_filter_type, [in] BOOL b_force, [out,
    retval] IDispatch** ppISimpleFilter)
```

Returns simple filter interface of the given filter type.

#### Parameters:

e_filter_type	enum that can have following values:
	FILTER_PERRET_OS_SKIP (1)
	FILTER_DLLP_UPDATE_FC (2)
	FILTER_LS_LINK_EVENT (3)
	FILTER_LS_WAKE (4)
	FILTER_LS_CLKREQ (5)
	FILTER_LS_PERST (6)

b\_force - boolean that can be 1 or 0 (1 is for enforcing even if the advanced trigger already exists. In this case the trigger will be converted to the simple type);

#### Return values

SimpleFilter object

#### Remarks

#### Example

```
Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set recOpt = Analyzer.GetRecordingOptions()
recOpt.SetRecMode(2)
ENFORCE = 1
FILTER_PERRET_OS_SKIP = 1
Set simpleFilter = recOpt.GetSimplefilter(FILTER_PERRET_OS_SKIP, ENFORCE)
```

See [7.6 SimpleFilter](#) for more information.

## 7.4.4 IPERecOptions3:: RestoreDefaultFactorySettings

```
HRESULT RestoreDefaultFactorySettings ()
```

Restores factory default settings.

**Parameters:** None

**Return Values:** None

### Example

```
Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set recOpt = Analyzer.GetRecordingOptions()
recOpt.RestoreDefaultFactorySettings()
```

## 7.5 SimpleTrigger

SimpleTrigger allows enabling /disabling the trigger and setting data stream direction.

The SimpleTrigger interface implements following methods:

*SetEnabled*  
*IsEnabled*  
*SetDirection*  
*GetDirection*

### 7.5.1 SimpleTrigger::SetEnabled

```
HRESULT SetEnabled ( [in] BOOL b_enabled )
```

Puts the trigger into enabled or disabled state.

### 7.5.2 SimpleTrigger::IsEnabled

```
HRESULT IsEnabled ( [out, retval] BOOL* pb_enabled )
```

Retrieves trigger's state.

### 7.5.3 SimpleTrigger::SetDirection

```
HRESULT SetDirection ( [in] EDirection e_dir )
```

Sets the trigger's direction.

#### Parameters:

e\_dir - enum that can have following values:

DIRECTION_UPSTREAM	( 0 )
DIRECTION_DOWNSTREAM	( 1 )
DIRECTION_BOTH	( 2 )
DIRECTION_NONE	( 4 )

#### Example

```
ENFORCE = 1
TRIGGER_EVENTTYPE_TS2 = 102
DIRECTION_UPSTREAM = 0
STATE_ENABLED = 1

Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set recOpt = Analyzer.GetRecordingOptions()
recOpt.SetRecMode(2)
Set simpleTrigger = recOpt.GetSimpleTrigger(TRIGGER_EVENTTYPE_TS2, ENFORCE)
simpleTrigger.SetEnabled(STATE_ENABLED)
simpleTrigger.SetDirection(DIRECTION_UPSTREAM)
```

## 7.5.4 SimpleTrigger::GetDirection

```
HRESULT GetDirection ( [out, retval] EDirection* pe_dir )
```

Gets the trigger's direction.

## 7.6 SimpleFilter

SimpleFilter allows enabling /disabling the filter and setting data stream direction..

The SimpleFilter interface implements following methods *SetEnabled*

*SetEnabled*  
*IsEnabled*  
*SetDirection*  
*GetDirection*

### 7.6.1 SimpleFilter::SetEnabled

```
HRESULT SetEnabled ( [in] BOOL b_enabled )
```

Puts the filter into enabled or disabled state.

### 7.6.2 SimpleFilter::IsEnabled

```
HRESULT IsEnabled ( [out, retval] BOOL* pb_enabled )
```

Retrieves filter's state.

### 7.6.3 SimpleFilter::SetDirection

```
HRESULT SetDirection ( [in] EDirection e_dir )
```

Sets the filter's direction.

#### Parameters:

e\_dir - enum that can have following values:

DIRECTION_UPSTREAM	( 0 )
DIRECTION_DOWNSTREAM	( 1 )
DIRECTION_BOTH	( 2 )
DIRECTION_NONE	( 4 )

#### Example

```
ENFORCE = 1
FILTER_PERRET_OS_SKIP = 1
DIRECTION_DOWNSTREAM = 1
STATE_ENABLED = 1

Set Analyzer = WScript.CreateObject("CATC.PETracer")
Set recOpt = Analyzer.GetRecordingOptions()
recOpt.SetRecMode(2)
Set simplefilter = recOpt.GetSimpleFilter(FILTER_PERRET_OS_SKIP, ENFORCE)
simplefilter.SetEnabled(STATE_ENABLED)
simplefilter.SetDirection(DIRECTION_DOWNSTREAM)
```

## 7.6.4 SimpleFilter::GetDirection

```
HRESULT GetDirection ( [out, retval] EDirection* pe_dir )
```

Gets the filter's direction.

## 7.7 Script Mode Automation API

Recording rules script can be saved to file or loaded from file using automation API. It is necessary to create PERecOptions object, which provides two COM methods. PERecOptions object can be created using GetRecordingOptions or GetRecordingOptionsEx COM methods.

```
HRESULT SaveRecOptionsScript([in] BSTR script_file_name)
```

COM method accepts path to the file, in which recording rules script will be saved. It converts recording rules to script text and saves to the specified file.

```
HRESULT LoadRecOptionsScript([in] BSTR script_file_name)
```

COM method accepts path to the file, which contains recording rules script text. It loads script from specified file, parses and converts to recording rules.

If script contains parse errors, then exception will be thrown, which will contain string with information about errors (line number and description of each error).

Examples:

Python:

```
from win32com.client import Dispatch
import pythoncom

analyzer = Dispatch("CATC.PETracer") # get PEInstance
rec_options = analyzer.GetRecordingOptionsEx(0)
try:
    rec_options.LoadRecOptionsScript(script_file_path)
except pythoncom.com_error as e:
    print(e.excepthook[2]) # Print string with information about errors
```

C++:

```
IPERecOptions* pe_rec_options;

...

try
{
    pe_rec_options->LoadRecOptionsScript(script_file_name);
}
catch (_com_error& e)
{
    ::MessageBox(NULL, e.Description(), _T("PETracer client"), MB_OK);
    // e.Description() - String with information about errors
}
```

If there are connected devices, then to use these methods at least one M5X device must be connected. These methods will return E\_FAIL error code if any error occurs.



## 8 PEGenOptions Object

The *PEGenOptions* object represents the options for the Summit Exerciser hardware. Also used to specify the traffic generation parameters.

The *PEGenOptions* object allows user to:

- Load/save the generation options from/to the file
- Set up custom generation parameters, such as link width, etc.

The *PEGenOptions* object can be created by using *IPEAnalyzer::GetGenerationOptions* method (see Page **Error! Bookmark not defined.**)

The *PEGenOptions* object supports the following interfaces:

Interfaces	Description
<i>IGenOptions</i>	Allows you to load/save recording options from/to the file and reset generation options
<i>IPEGenOptions</i>	Identical to <i>IGenOptions</i> interface
<i>IPEGenOptions2</i>	Extends the <i>IPEGenOptions</i> interface. Adds set up for link width, spec mode, external reference clock, descrambling algorithm, skew, lane reversal, and polarity inversion

The *IPERecOptions2* interface is a primary interface for *PERecOptions* object.

## 8.1 IGenOptions interface

The *IGenOptions* interface is a dual interface for the *PEGenOptions* object.

*IGenOptions* implements the following methods:

- Load*
- Save*
- Reset*

**Note:** All methods of the *IGenOptions* interface are also available in the *IPEGenOptions* (see Page 73) and the *IPEGenOptions2* (see Page 73) interfaces.

### 8.1.1 IGenOptions::Load

```
HRESULT Load (
    [in] BSTR file_name )
```

Loads generation options from the specified file.

#### Parameters

file_name	String that provides the full pathname to the generation options file
-----------	---

#### Return values

ANALYZERCOMERROR_UNABLEOPENFILE	Unable to open file
---------------------------------	---------------------

#### Remarks

#### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions
GenOptions.Load( CurrentDir & "Input\gen_options.gen" )
```

C++:

## 8.1.2 IGenOptions::Save

```
HRESULT Save (  
    [in] BSTR file_name )
```

Saves generation options into the specified file.

### Parameters

**file\_name**                      String that provides the full pathname to the generation options file

### Return values

**ANALYZERCOMERROR\_UNABLEOPENFILE**              Unable to open file

### Remarks

If the specified file does not exist, it is created; if it exists, it is overwritten.

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )  
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set GenOptions = Analyzer.GetGenerationOptions  
GenOptions.Save( CurrentDir & "Input\gen_options.gen" )
```

C++:

### 8.1.3 IGenOptions::Reset

```
HRESULT Reset ( )
```

Resets the generation options to its initial state.

#### Parameters

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set GenOptions = Analyzer.GetGenerationOptions( )  
GenOptions.Reset( )
```

C++:

## 8.2 IPEGenOptions interface

This interface is identical to the *IGenOptions* interface (see Page 69).

## 8.3 IPEGenOptions2 interface

The *IPEGenOptions2* interface is a primary dual interface for the *PEGenOptions* object.

This interface is derived from the *IPEGenOptions* interface.

The *IPEGenOptions2* interface implements all methods from the *IPEGenOptions* interface, plus the following:

- SetTargetGenerator*
- SetWorkAsRoot*
- SetLinkWidth*
- SetBase10Spec*
- SetExternalRefClock*
- SetDisableDescrambling*
- SetDisableScrambling*
- SetAutoConfig*
- SetReverseLanes*
- SetInvertPolarity*
- SetSkew*

### 8.3.1 IPEGenOptions2::SetTargetGenerator

```
HRESULT SetTargetGenerator (
    [in] ETargetGenerator target_generator )
```

Sets the hardware configuration for the generation options.

#### Parameters:

target_generator	Enumerated value that provides the platform to set; <i>ETargetGenerator</i> has the following values:
	TARGETGENERATOR_Z3      (3) - Summit Z3
	TARGETGENERATOR_Z4      (4) - Summit Z416

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions( )
GenOptions.SetTargetGenerator( 1 )
```

C++:

### 8.3.2 IPEGenOptions2::SetWorkAsRoot

```
HRESULT SetWorkAsRoot  
    ( [in] BOOL root )
```

#### Parameters

root	If TRUE, then the Summit Exerciser emulates a Root Complex, If FALSE, it emulates an endpoint device.
------	--

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set GenOptions = Analyzer.GetGenerationOptions( )  
GenOptions.SetWorkAsRoot( 0 )
```

C++:

### 8.3.3 IPEGenOptions2::SetLinkWidth

```
HRESULT SetLinkWidth (
    [in] int link_width )
```

Sets the link width.

#### Parameters:

link_width	Link width to set; allowed values are:
	Summit Z3      1, 2, 4, 8, 16
	Summit Z416    1, 2, 4, 8, 16

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions
GenOptions.SetLinkWidth( 1 )
```

C++:

### 8.3.4 IPEGenOptions2::SetBase10Spec

```
HRESULT SetBase10Spec (  
    [in] BOOL base_10_spec )
```

Sets the PCI Express Base Specification 1.0 compatibility mode.

#### Parameters

base_10_spec	When TRUE, the Summit Exerciser hardware uses Base Spec 1.0 compatibility mode
--------------	--

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set GenOptions = Analyzer.GetGenerationOptions  
GenOptions.SetBase10Spec( 0 )
```

C++:

### 8.3.5 IPEGenOptions2::SetExternalRefClock

```
HRESULT SetExternalRefClock(  
    [in] BOOL ext_ref_clock )
```

Specifies whether to use the external or the internal reference clock

#### Parameters

ext\_ref\_clock            When TRUE, the external reference clock is used

#### Remarks

Not implemented for Summit Z3.

### 8.3.6 IPEGenOptions2::SetDisableDescrambling

```
HRESULT SetDisableDescrambling (
    [in] BOOL disable_descrambling )
```

Disables/enables descrambling of incoming traffic.

#### Parameters

disable_descrambling	When TRUE, descrambling is disabled
----------------------	-------------------------------------

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions( )
GenOptions.SetDisableDescrambling( 0 )
```

C++:

### 8.3.7 IPEGenOptions2::SetDisableScrambling

```
HRESULT SetDisableScrambling (
    [in] BOOL disable_scrambling )
```

Disables/enables scrambling of outgoing traffic.

#### Parameters

disable_scrambling	When TRUE, scrambling is disabled
--------------------	-----------------------------------

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions
GenOptions.SetDisableScrambling( 0 )
```

C++:

### 8.3.8 IPEGenOptions2::SetAutoConfig

```
HRESULT SetAutoConfig (
    [in] BOOL auto_config )
```

Enables/disables automatic link configuration detection.

#### Parameters

auto_config	When TRUE, the following parameters of the generation options are detected automatically: <ul style="list-style-type: none"><li>• Link width</li><li>• Scrambling of outgoing traffic</li><li>• Descrambling of incoming traffic</li><li>• Lane reversal</li><li>• Polarity inversion of incoming traffic</li></ul>
-------------	---

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions( )
GenOptions.SetAutoConfig( 0 )
```

C++:

### 8.3.9 IPEGenOptions2::SetReverseLanes

```
HRESULT SetReverseLanes (
    [in] EDirection direction,
    [in] BOOL reverse )
```

Allows lane reversal in the specified traffic direction.

#### Parameters:

direction	Enumerated value that provides traffic direction for lane reversal; <i>EDirection</i> has the following values: DIRECTION_UPSTREAM (0) - upstream traffic DIRECTION_DOWNSTREAM (1) - downstream traffic
reverse	Specifies whether to reverse the lanes of the specified link direction

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions
GenOptions.SetReverseLanes( 0, 1 )    ` reverse lanes in upstream traffic
GenOptions.SetReverseLanes( 1, 1 )    ` reverse lanes in downstream traffic
```

C++:

### 8.3.10 IPEGenOptions2::SetInvertPolarity

```
HRESULT SetInvertPolarity (
    [in] EDirection direction,
    [in] int lane,
    [in] BOOL invert )
```

Allows polarity inversion on the specified lane and specified traffic direction.

#### Parameters:

direction	Enumerated value that provides traffic direction for polarity inversion; <i>EDirection</i> has the following values: DIRECTION_UPSTREAM (0) - upstream traffic DIRECTION_DOWNSTREAM (1) - downstream traffic
lane	Specifies the lane to invert polarity on
invert	Sets polarity inversion on the specified lane of the specified link direction

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions( )

For i = 1 To 4
    GenOptions.SetInvertPolarity( 0, i, 0 )
    GenOptions.SetInvertPolarity( 0, i, 1 )
Next
```

C++:

### 8.3.11 IPEGenOptions2::SetSkew

```
HRESULT SetSkew (
    [in] int lane,
    [in] int skew )
```

Allows skew values to be set for each lane of outgoing traffic.

#### Parameters

lane	Specifies the lane to set the skew value on
skew	Specifies the numeric value for the skew on the specified lane; allowed values are from 0 to 7

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set GenOptions = Analyzer.GetGenerationOptions

GenOptions.SetSkew( 0, 0 )    ` set skew value 0 for lane 0
GenOptions.SetSkew( 1, 2 )    ` set skew value 2 for lane 1
GenOptions.SetSkew( 2, 0 )    ` set skew value 0 for lane 2
GenOptions.SetSkew( 3, 3 )    ` set skew value 3 for lane 3
```

C++:

## 9 PEPacket Object

The *PEPacket* object represents a single packet of the recorded trace file.

The *PEPacket* object allows user to retrieve packet content and packet properties such as timestamp, link width, packet start lane, packet direction, and packet errors.

The *PEPacket* object can be created by calling *IPETrace::GetBusPacket* method (See Page 41)

The *PEPacket* object supports the following interfaces:

Interfaces	Description
<i>IPacket</i>	Allows retrieval of the packet's timestamp
<i>IPEPacket</i>	Extends the <i>IPacket</i> interface

The *IPEPacket* interface is a primary interface for the *PEPacket* object.

## 9.1 IPacket interface

The *IPacket* interface is a dual interface for *PEPacket* object.

*IPacket* implements the following method:

*GetTimestamp*

**Note:** All methods of the *IPacket* interface are also available in the *IPEPacket* interface (see Page 57).

### 9.1.1 IPacket::GetTimestamp

```
HRESULT GetTimestamp (
    [out, retval] double* timestamp )
```

Returns the packet timestamp in nanoseconds.

#### Parameters

timestamp	Timestamp of the beginning symbol of the packet from the start of recording
-----------	---

#### Return values

#### Remarks

#### Example

WSH:

```
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
TriggerPacket = Trace. GetTriggerPacketNum
Set Packet = Trace.GetBusPacket(TriggerPacket)
MsgBox "Trigger packet at " & Packet.GetTimestamp & " ns"
```

C++:

## 9.2 IPEPacket interface

The *IPEPacket* interface is a primary dual interface for the *PEPacket* object.

This interface is derived from the *IPacket* interface.

The *IPEPacket* interface implements all methods from the *IPacket* interface plus the following:

- GetPacketData*

- GetLinkWidth*

- GetStartLane*

- GetLFSR*

- GetDirection*

- GetErrors*

## 9.2.1 IPEPacket::GetPacketData

```
HRESULT GetPacketData (
    [in] EPacketFormat format,
    [out] VARIANT* packet,
    [out, retval] long* number_of_bytes )
```

Retrieves a raw packet representation.

### Parameters

format	Data representation format; the <i>EPacketFormat</i> enumerator has the following values:						
	<table> <tr> <td>PACKETFORMAT_BYTES</td> <td>( 0 ) bytes</td> </tr> <tr> <td>PACKETFORMAT_SCRAMBLED_BYTES</td> <td>( 1 ) scrambled bytes</td> </tr> <tr> <td>PACKETFORMAT_TEN_BIT</td> <td>( 2 ) 10bit codes</td> </tr> </table>	PACKETFORMAT_BYTES	( 0 ) bytes	PACKETFORMAT_SCRAMBLED_BYTES	( 1 ) scrambled bytes	PACKETFORMAT_TEN_BIT	( 2 ) 10bit codes
PACKETFORMAT_BYTES	( 0 ) bytes						
PACKETFORMAT_SCRAMBLED_BYTES	( 1 ) scrambled bytes						
PACKETFORMAT_TEN_BIT	( 2 ) 10bit codes						
packet	Raw packet data						
number_of_bytes	Number of bytes in the packet						

### Return values

ANALYZERCOMERROR_WRONGCALL	Unknown packet format specified
----------------------------	---------------------------------

### Remarks

*packet* parameter has *VT\_ARRAY|VT\_VARIANT* actual automation type. For *PACKETFORMAT\_BYTES* and *PACKETFORMAT\_SCRAMBLED\_BYTES*, each element of this array has the *VT\_UI1* automation type. For *PACKETFORMAT\_TEN\_BIT*, each element of this array has the *VT\_UI2* automation type.

## Example

VBScript:

```

<OBJECT
  ID = Analyzer
  CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>
<INPUT TYPE=TEXT NAME="TextPacketNumber">
<P ALIGN=LEFT ID=StatusText></P>

<SCRIPT LANGUAGE="VBScript">
<!--
Function DecToBin(Param, NeedLen)
  While Param > 0
    Param = Param/2
    If Param - Int(Param) > 0 Then
      Res = CStr(1) + Res
    Else
      Res = CStr(0) + Res
    End If
    Param = Int(Param)
  Wend
  DecToBin = Replace( Space(NeedLen - Len(Res)), " ", "0") & Res
End Function

Sub BtnGetPacket_OnClick
  ClearStatus()
  On Error Resume Next
  Set Packet = CurrentTrace.GetBusPacket (TextPacketNumber.value)

  If Err.Number <> 0 Then
    MsgBox "GetBusPacket:" & Err.Number & ":" & Err.Description
  Else
    Timestamp = Packet.GetTimestamp()
    If Err.Number <> 0 Then
      MsgBox "GetTimestamp:" & Err.Number & ":" & Err.Description
    End If

    NumberOfUnits = Packet.GetPacketData ( PACKETFORMAT_BYTES, PacketData)

    If Err.Number <> 0 Then
      MsgBox "GetPacketData:" & Err.Number & ":" & Err.Description
    Else

      For Each PacketByte In PacketData
        PacketStr = PacketStr & DecToBin(PacketByte, 8) & " "
        NBytes = NBytes + 1
      Next

      StatusText.innerText = "Packet ( " & NumberOfUnits & " bytes ): " & PacketStr
    End If
  End If
End Sub
-->
</SCRIPT>

```

```

C++:
IPEPacket* custom_packet;
LONG packet_number;

. . .

VARIANT packet_data;
double timestamp_ns;
VariantInit( &packet_data );
long number_of_bytes;
try
{
    number_of_bytes = custom_packet->GetPacketData( PACKETFORMAT_BYTES, &packet_data );
    timestamp_ns    = custom_packet->GetTimestamp ( );
}
catch ( _com_error& er)
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(),_T("PETracer client"), MB_OK );
    return 1;
}

if ( packet_data.vt == ( VT_ARRAY | VT_VARIANT) )
{
    SAFEARRAY* packet_safearray = packet_data.parray;

    TCHAR* packet_message = new TCHAR [ 3*packet_safearray->rgsabound[0].cElements + 64 ];
    TCHAR elem[64];
    _stprintf( packet_message, _T("packet #%ld: "), GetDlgItemInt(IDC_PACKET_NUMBER) );

    _stprintf( elem, _T(" %.0lf ns"), timestamp_ns );
    _tcscat( packet_message, elem );

    _stprintf( elem, _T(", %d bytes:  "), number_of_bytes );
    _tcscat( packet_message, elem );

    for ( long i=0; i<(long)packet_safearray->rgsabound[0].cElements; i++)
    {
        VARIANT var;
        HRESULT hr = SafeArrayGetElement(packet_safearray, &i, &var);
        if (FAILED(hr))
        {
            ::MessageBox( NULL, _T("Error accessing array"), _T("PETracer client"), MB_OK );
            return 1;
        }
        if ( var.vt != ( VT_UI1 ) )
        {
            ::MessageBox( NULL, _T("Array of bytes expected"), _T("PETracer client"), MB_OK );
            return 1;
        }

        _stprintf( elem, _T("%02X "), V_UI1(&var) );
        _tcscat( packet_message, elem );

    }

    ::MessageBox( NULL, packet_message, _T("packet"), MB_OK );

    delete [] packet_message;
}
else
{
    ::MessageBox( NULL, _T("Invalid argument"), _T("PETracer client"), MB_OK );
}

```

## 9.2.2 IPEPacket::GetLinkWidth

```
HRESULT GetLinkWidth (
    [out, retval] long* width )
```

Returns link width for this packet.

### Parameters

width	Link width for the packet
-------	---------------------------

### Return values

### Remarks

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.OpenFile( CurrentDir & "Input\errors.pex" )
Set Packet = Trace.GetBusPacket( 0 )
MsgBox "Link width: " & Packet.GetLinkWidth
```

C++:

### 9.2.3 IPEPacket::GetStartLane

```
HRESULT GetStartLane (
    [out, retval] long* start_lane )
```

Returns start lane for this packet.

#### Parameters

start\_lane                      Start lane for the packet

#### Return values

#### Remarks

#### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.OpenFile( CurrentDir & "Input\errors.pex" )
Set Packet = Trace.GetBusPacket( 0 )
MsgBox "Start lane: " & Packet.GetStartLane
```

C++:

## 9.2.4 IPEPacket::GetDirection

```
HRESULT GetDirection (
    [out, retval] long* direction )
```

Returns direction (upstream/downstream) of this packet.

### Parameters

direction	0 – upstream packet
	1 – downstream packet

### Return values

### Remarks

### Example

WSH:

```
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )
Set Trace = Analyzer.OpenFile( CurrentDir & "Input\errors.pex" )
Set Packet = Trace.GetBusPacket( 0 )
MsgBox "Direction: " & Packet.GetDirection
```

C++:

## 9.2.5 IPEPacket::GetErrors

```
HRESULT GetErrors (
    [out] VARIANT* error_array,
    [out, retval] long* number_of_errors )
```

Returns an array of errors present in this packet.

### Parameters

error_array	Array of error id present in this packet. See <i>ITrace::AnalyzerErrors</i> , Page 39, for error id values
number_of_errors	Total number of errors in this packet

### Return values

### Remarks

### Example

WSH:

C++:

## 10 PETraceErrors Object

The *PETraceErrors* object represents the collection of errors that occurred in the recorded trace file.

The *PETraceErrors* object can be created by calling *ITrace::AnalyzerErrors* method (see Page **Error! Bookmark not defined.**).

The *IAnalyzerErrors* interface is a primary interface for the *PETraceErrors* object.

### 10.1 IAnalyzerErrors dispinterface

This is a standard collection interface for collection of packet numbers with errors of a specified type (see *ITrace::AnalyzerErrors*, Page 39).

It has the following methods, which are standard for the collection interfaces:

- get\_Item*
- get\_Count*

### 10.1.1 IAnalyzerErrors::get\_Item

```
HRESULT get_Item(  
    [in] long index,  
    [out, retval] long* packet_number )
```

Returns a zero based packet number from error collection

#### Parameters

index	Index of the error in the collection
packet_number	Error packet number

## 10.1.2 IAnalyzerErrors::get\_Count

```
HRESULT get_Count(  
    [out, retval] long* number_of_errors )
```

Returns the number of errors in the trace.

### Parameters

number_of_errors	Number of elements in the collection
------------------	--------------------------------------

### Remarks

### Example

WSH:

```
' makes recording, saves the portions of the recorded trace  
' where "Running Disparity" errors ocured  
CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, "\" ) )  
Set Analyzer = WScript.CreateObject( "CATC.PETracer" )  
Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )  
Set Errors = Trace.AnalyzerErrors( 32 ) ' Running Disparity Error  
For Each ErrorPacketNumber In Errors  
    ErrorFile = CurrentDir & "\Output\PckLen_error_span_" &  
        CStr(ErrorPacketNumber) & ".pex"  
    Trace.Save ErrorFile, CInt(ErrorPacketNumber)-5, CInt(ErrorPacketNumber)+5  
Next
```

C++:

```

IPETTrace* pe_trace;

. . .

IAnalyzerErrors* analyser_errors;
try
{
    analyser_errors = pe_trace->AnalyzerErrors(error_type).Detach();
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETTracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETTracer client"), MB_OK );
    return 1;
}

TCHAR all_errors[2048];
_stprintf( all_errors, _T("Errors: ") );
try
{
    long errors_count = analyser_errors->GetCount();
    long analyzer_error;
    if ( !errors_count )
    {
        _tcscat( all_errors, _T("none") );
    }
    for ( long i=0; i<errors_count && i<2048/32; i++ )
    {
        analyzer_error = analyser_errors->GetItem(i);
        TCHAR cur_error[32];
        _stprintf( cur_error, _T(" %ld"), analyzer_error );
        _tcscat( all_errors, cur_error );
    }
    if ( i>2048/32 )
        _tcscat( all_errors, _T(" ...") );
}
catch ( _com_error& er )
{
    if (er.Description().length() > 0)
        ::MessageBox( NULL, er.Description(), _T("PETTracer client"), MB_OK );
    else
        ::MessageBox( NULL, er.ErrorMessage(), _T("PETTracer client"), MB_OK );
    return 1;
}

analyser_errors->Release();

::SetWindowText( m_hwndStatus, all_errors );

```

## 11 PEVScriptEngine Object

The *PEVScriptEngine* object allows a user to run verification scripts over the recorded trace. It extends the functionality of the *IPEVerificationScript* interface of a *PETrace* object. The main advantage of a *PEVScriptEngine* object is that it allows clients implementing *\_IVScriptEngineEvents* a callback interface to receive notifications when a verification script is running.

The *PEVScriptEngine* object can be created by calling *IPEVerificationScript::GetVScriptEngine* method (see Page 45).

The *PEVScriptEngine* object supports the following interfaces:

Interfaces	Description
<i>IVScriptEngine</i>	Provides advanced control over the verification script and allows you to execute the script asynchronously
<i>IAnalyzerEvents</i>	Events from <i>PEVScriptEngine</i> object

The *IVScriptEngine* interface is a primary interface for *PEVScriptEngine* object.

### Remarks

Verification scripts are scripts written in a special manner using the *CATC Script Language (CSL)*. These scripts can be “run” over a recorded trace to “verify” the trace for some verification conditions or to extract more advanced information from the trace. Such scripts utilize a special feature of the PCIe Protocol Analysis™ application, its *Verification Script Engine*.

Please refer to the *PCIe Protocol Analysis Manual*, the *PCIe Protocol Analysis Verification Script Engine Manual*, and the *PCIe Protocol Analysis File Based Decoding Manual* for more details.

## 11.1 IVScriptEngine interface

The *IVScriptEngine* interface is the primary dual interface for the *PEVScriptEngine* object.

It implements the following properties and methods:

- VscriptName*
- Tag*
- RunVScript*
- RunVScriptEx*
- LaunchVScript*
- Stop*
- GetScriptVar*
- SetScriptVar*

### 11.1.1 IVScriptEngine::VScriptName

```
[propget] HRESULT VScriptName( [out, retval] BSTR *pVal )
[propput] HRESULT VScriptName( [in] BSTR newVal )
```

Property putting and getting current verification script name.

#### Parameters

pVal	Address of the variable where the current verification script name is kept
newVal	Name of the verification script to initialize script verification engine

#### Return values

#### Remarks

The name of verification script is the name of verification script file (\*.pevs). If only the name of the script without file extension is specified, the PCIe Protocol Analysis server is going to search for the named script among the scripts loaded from the \Scripts\VFScripts folder under PCIe Protocol Analysis installation folder. If the full path to the script is specified, then the server is going to attempt loading the script from the specified path prior to running it.

#### Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vscript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vscript )
assert( pe_vscript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vscript -> GetVScriptEngine("MyVSEngine");
assert( pe_vsengine != NULL );

pe_vsengine -> PutVScriptName("Test_1");
assert( pe_vsengine -> GetVScriptName() == "Test_1" );

VS_RESULT result = pe_vsengine ->RunVScript();
if( result == PASSED )
{
    ::MessageBox( NULL, "Test 1 passed !!!", "PETracer client", MB_OK );
}

. . .
```

## 11.1.2 IVScriptEngine::Tag

```
[propget] HRESULT Tag( [out, retval] int* pVal )
[propput] HRESULT Tag( [in] int newVal )
```

Property assigning and getting a tag to the VSE object. This tag is used in event notifications allowing a client event handler to determine which VSE object sent the event.

### Parameters

pVal	Address of the variable where the current VSE tag is kept
newVal	New tag for VSE

### Return values

### Remarks

### Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vsript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vsript )
assert( pe_vsript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vsript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

pe_vsengine ->PutTag( 0xDDAADDAA );
assert( pe_vsengine -> GetTag() == 0xDDAADDAA );

VS_RESULT result = pe_vsengine ->RunVScript();
if( result == PASSED )
{
    ::MessageBox( NULL, "Test 1 passed !!!", "PETracer client", MB_OK );
}

. . .
```

### 11.1.3 IVScriptEngine::RunVScript

```
HRESULT RunVScript( [out, retval] int* pResult )
```

Runs the verification script currently specified for this engine.

#### Parameters

**pResult**                      Address of a variable where the results of the verification is kept.

#### Return values

#### Remarks

This method makes a “synchronous” call – which means that this method doesn’t return until the script stops running. See *IPEVerificationScript::RunVerificationScript* method, Page 43, for details.

#### Example

See C++ example to *IVScriptEngine::VScriptName*, Page 101.

### 11.1.4 IVerScriptEngine::RunVScriptEx

```
HRESULT RunVScriptEx(  
    [in] BSTR script_name,  
    [out, retval] int* pResult )
```

Changes the current verification script name and runs verification script.

#### Parameters

script_name	Name of the verification script to initialize the script verification engine
pResult	Address of a variable where the results of a verification is kept

#### Return values

#### Remarks

This method makes a “synchronous” call – which means that this method doesn’t return until the script stops running.

The name of verification script is the name of verification script file (\*.pevs). If only the name of the script without file extension is specified, the PCIe Protocol Analysis server is going to search for the named script among the scripts loaded from the \Scripts\VFScripts folder under PCIe Protocol Analysis installation folder. If the full path to the script is specified, then the server is going to attempt loading the script from the specified path prior to running it. See *IPEVerificationScript::RunVerificationScript* method, Page 43, for details.

## Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vsript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vsript )
assert( pe_vsript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vsript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

VS_RESULT result = pe_vsengine ->RunVScript();
if( result == PASSED )
{
    ::MessageBox( NULL, "Test 1 passed !!!", "PETracer client", MB_OK );
}

result = pe_vsengine ->RunVScriptEx("Test_2");
if( result == PASSED )
{
    ::MessageBox( NULL, "Test 2 passed !!!", "PETracer client", MB_OK );
}

result = pe_vsengine ->RunVScriptEx("C:\\MyTests\\Test_3.pevs");
if( result == PASSED )
{
    ::MessageBox( NULL, "Test 3 passed !!!", "PETracer client", MB_OK );
}

. . .
```

## 11.1.5 IVScriptEngine::LaunchVScript

```
HRESULT LaunchVScript()
```

Launches verification script.

### Return values

`S_FALSE` If VS Engine was not successfully launched  
(either it is already running or verification script was not found)

### Remarks

This method makes an “asynchronous” call, which means that this method immediately returns after the script starts running.

When the verification script stops running, the VSE object sends a special event notification *OnVScriptFinished* (see Page 116) to the client event handler. You can also terminate the running script using the method *Stop* (Page 107).

### Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vs_script = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vs_script )
assert( pe_vs_script != NULL );

IVScriptEngine* pe_vs_engine = NULL;
pe_vs_engine = pe_vs_script -> GetVScriptEngine("Test_1");
assert( pe_vs_engine != NULL );

VS_RESULT result = pe_vs_engine ->LaunchVScript();

// You can go further without waiting the result from the VSE object.
// If you interested in the result you should implement the client event handler for
// OnVScriptFinished() notification.
. . .
```

### 11.1.6 IVScriptEngine::Stop

```
HRESULT Stop()
```

Stops verification script previously launched by the *IVScriptEngine::LaunchVScript* method (see Page 106).

#### Parameters

#### Return values

#### Remarks

#### Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vscript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vscript )
assert( pe_vscript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vscript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

VS_RESULT result = pe_vsengine ->LaunchVScript();
. . .

if( NotEnoughResourcesToProcessVS )
    pe_vsengine ->Stop();
. . .
```

### 11.1.7 IVScriptEngine::GetScriptVar

```
HRESULT GetScriptVar (
    [in] BSTR var_name,
    [out, retval] VARIANT* var_value )
```

Returns the value of some verification script global variables before/after executing the script (refer to the *PCIe Protocol Analysis Verification Script Engine Manual* and the *File Based Decoding Manual* for information on how a script can declare and set global variables). The resulting value may contain an integer, a string, or an array of VARIANTS (if a requested script variable is a list object – see the *PCIe Protocol Analysis File Based Decoding Manual* for more details about list objects)

#### Parameters

<code>var_name</code>	String providing the name of the global variable or constant used in the verification script running
<code>var_value</code>	Address of a VARIANT variable where the result is kept

#### Return values

<code>E_PENDING</code>	If this method is called when the script is already running
------------------------	---

#### Remarks

If there is no such global variable or constant with the name `var_name`, the resulting value contains an empty VARIANT.

## Example

C++:

```
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;

. . .

IPEVerificationScript* pe_vscript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void**)&pe_vscript )
assert( pe_vscript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vscript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

VS_RESULT result = pe_vsengine ->RunVScript();
. . .

VARIANT my_var;
VariantInit( &my_var );

pe_vsengine->GetScriptVar( _bstr_t("MyVar"), &my_var );

if( my_var.vt == VT_BSTR ) ProcessString( my_var.bstrVal );
. . .
```

WSH:

```
. . .
Set Trace      = Analyzer.OpenFile( TraceName )      ' Open the trace
Set VSEngine = Trace.GetVScriptEngine( VScript ) ' Get VS Engine object

Result = VSEngine.RunVScript

integer MyIntVar = VSEngine.GetScriptVar( "MyIntVar" ) ' Let's suppose that MyIntVar contains an
string  MyStrVar = VSEngine.GetScriptVar( "MyStrVar" ) ' Let's suppose that MyStrVar contains a

MsgBox " MyIntVar = " & CStr(MyIntVar) & ", MyStrVar = " & MyStrVar
```

## 11.1.8 IVScriptEngine::SetScriptVar

```
HRESULT SetScriptVar ( [in] BSTR var_name, [in] VARIANT var_value )
```

This method allows you to set the value of some verification script global variable before/after executing the script (refer to the *PCIe Protocol Analysis Verification Script Engine Manual* and the *File Based Decoding Manual* for information on how a script can declare, set, and use global variables). Only integers, strings, or arrays of VARIANTS are allowed as correct values. Arrays of VARIANTS is converted into list values inside of scripts. See the *PCIe Protocol Analysis File Based Decoding Manual* for more details about list objects.

### Parameters

var_name	String providing the name of the global variable used in the verification script being run
var_value	VARIANT value containing the new variable value

### Return values

E_PENDING	If this method is called when the script is already running
-----------	---

### Remarks

This function may be very useful because it allows you to set internal script variables before running a script, giving you the opportunity to make run-time customization from COM/Automation client applications.

In order for this operation to take effect during execution of the script, a global variable with the name specified by *var\_name* should be declared by the script.

### Example

```
C++:
// In this example we use wrapper functions provided by #import directive
//
IPETrace* pe_trace;
. . .

IPEVerificationScript* pe_vscript = NULL;

pe_trace->QueryInterface( IID_IPEVerificationScript, (void*)&pe_vscript )
assert( pe_vscript != NULL );

IVScriptEngine* pe_vsengine = NULL;
pe_vsengine = pe_vscript -> GetVScriptEngine("Test_1");
assert( pe_vsengine != NULL );

VARIANT my_var;
VariantInit( &my_var );

my_var.vt = VT_I4; // Integer
my_var.lVal = 100;

// set internal script variable 'MyVar' to 100
pe_vsengine->SetScriptVar( _bstr_t("MyVar"), my_var );

VS_RESULT result = pe_vsengine ->RunVScript();
. . .
```

WSH:

```
. . .  
Set Trace      = Analyzer.OpenFile( TraceName )      ' Open the trace  
Set VSEngine = Trace.GetVScriptEngine( VScript ) ' Get VS Engine object  
  
VSEngine.GetScriptVar( "MyIntVar" , 100 )  
VSEngine.GetScriptVar( "MyStrVar" , "Hello !!!" )  
Result = VSEngine.RunVScript
```

## 12 PEVScriptEngine Object Events

### 12.1 \_IVScriptEngineEvents interface

In order to retrieve the event notifications from PCIe Protocol Analysis™ application when a verification script engine object is running the script, you must implement the *\_IVScriptEngineEvents* callback interface. Since this interface is a default source interface for the *PEVScriptEngine* object, there is a very simple implementation from such languages like Visual Basic, VBA, VBScript, WSH, etc.

Some script engines impose restrictions on handling events from “indirect” automation objects in typeless script languages (when an automation interface to the object is obtained from a call of some method rather than from creation function – like *CreateObject()* in VBScript). The PCIe Protocol Analysis provides a special COM class allowing the receiving and handling of notifications from a VSE object even in script languages not supporting event handling from “indirect” objects. Please refer to *CATCAnalyzerAdapter*, Page 123, for details.

#### Example

C++:

C++ implementation used in the examples below implements an event sink object by deriving it from *IDispEventImpl*, but not specifying the type library as a template argument. Instead, the type library and default source interface for the object are determined using *AtlGetObjectSourceInterface()*. A *SINK\_ENTRY()* macro is used for each event from each source interface that is to be handled:

```
class CVSEngineSink : public IDispEventImpl<IDC_SRCOBJ_VSE, CVSEngineSink >
{
public:
...

BEGIN_SINK_MAP(CVSEngineSink)
    //Make sure the Event Handlers have __stdcall calling convention
    SINK_ENTRY( IDC_SRCOBJ_VSE, 1, OnVScriptReportUpdated )
    SINK_ENTRY( IDC_SRCOBJ_VSE, 2, OnVScriptFinished )
    SINK_ENTRY( IDC_SRCOBJ_VSE, 3, OnNotifyClient )
END_SINK_MAP()

HRESULT __stdcall OnVScriptReportUpdated ( BSTR newLine, int TAG );
HRESULT __stdcall OnVScriptFinished( BSTR script_name, VS_RESULT result, int TAG );
HRESULT __stdcall OnNotifyClient ( int eventId, VARIANT eventBody, int TAG );

HRESULT Advise(IUnknown* pUnk)
{
    AtlGetObjectSourceInterface(pUnk,
        &m_libid, &m_iid, &m_wMajorVerNum, &m_wMinorVerNum);
    return DispEventAdvise(pUnk, &m_iid);
}

HRESULT Unadvise(IUnknown* pUnk)
{
    AtlGetObjectSourceInterface(pUnk,
        &m_libid, &m_iid, &m_wMajorVerNum, &m_wMinorVerNum);
    return DispEventUnadvise(pUnk, &m_iid);
}

...
};
```

Then, after you've established the connection with the server, you need to advise your implementation of the event interface:

```
IVScriptEngine vscript_engine = NULL;

try
{
    vscript_engine = vscript ->GetVScriptEngine( "Test_1" );
}
catch ( _com_error& er )
{
    SetStatusError( er );
}

if ( vscript_engine == NULL )
{
    vscript = NULL;
    return E_FAIL;
}

CVSEngineSink vse_sink;
HRESULT hr = vse_sink . Advise( vscript_engine ); // "Subscribe" for receiving events

...

VS_RESULT res = SCRIPT_NOT_FOUND;
try
{
    res = (VS_RESULT)vscript_engine ->RunVScript();
}
catch ( _com_error& er )
{
    SetStatusError( er );
}

// Tear connection with the test case
vse_sink.Unadvise( vscript_engine );

...
```

```

VBA: ( MS Excel )

Public PETracer As PeAnalyzer
Public Trace As PeTrace
Public GVSEngine As VScriptEngine

'
' VSEngineEventsModule - is a special class implementing VSE event handlers.
' It should have in global declaration section the line like this:
' Public WithEvents VSEEvents As VScriptEngine
'
Dim X As New VSEngineEventsModule...

Private Sub RunVScritButton_Click()
    Dim VSEngine As VScriptEngine
    Dim IVScript As IPEVerificationScript
    Dim ScriptName, fileToOpen As String

    ScriptName = ThisWorkbook.Sheets("Sheet1").Cells(2, 2)

    If PETracer Is Nothing Then
        Set PETracer = New PeAnalyzer

        If PETracer Is Nothing Then
            MsgBox "Unable to connect to PETracer", vbExclamation
            Exit Sub
        End If
    End If

    fileToOpen = ThisWorkbook.Sheets("Sheet1").Cells(1, 2)
    Set Trace = PETracer.OpenFile( fileToOpen )

    Set IVScript = Trace 'Get the IfcVerificationScript interface
    Set VSEngine = IVScript.GetVScriptEngine( ScriptName )

    ' "Subscribe" for receiving VSE events -
    ' the X variable ( an instance of VSEngineEventsModule class ) handles them.
    '
    Set X.VSEEvents = VSEngine

    ...

    VSEngine.Tag = 12 ' Assign a tag for VSE object
    VSEngine.RunVScript ' Run verification script

    Set X.VSEEvents = Nothing ' "Unsubscribe" for receiving VSE events
    Set VSEngine = Nothing ' Release external
    Set IVScript = Nothing ' objects...
End Sub

```

### 12.1.1 `_IVScriptEngineEvents::OnVScriptReportUpdated`

```
HRESULT OnVScriptReportUpdated (
    [in] BSTR newLine,
    [in] int TAG )
```

Fired when running a verification script, calls the *ReportText( newLine )* function (please refer to the *PCIe Protocol Analysis Verification Script Engine Manual* for details on the *ReportText* function).

#### Parameters

<code>newLine</code>	New portion of text reported by the verification script
<code>TAG</code>	VSE object's tag

#### Return values

#### Remarks

Make sure that C++ event handlers have `__stdcall` calling convention.

#### Example

C++:

```
HRESULT __stdcall OnVScriptReportUpdated (BSTR newLine, int TAG )
{
    TRACE( "Line: %s, TAG: %d\n", newLine, TAG );
    . . .

    return S_OK;
}
```

VBA (MS Excel):

```
Public WithEvents VSEEvents As VScriptEngine
Public LineIndex As Integer
. . .
Private Sub VSEEvents_OnVScriptReportUpdated(ByVal newLine As String, ByVal Tag As Long)

    ThisWorkbook.Sheets("Sheet1").Cells(LineIndex, 1) = newLine
    LineIndex = LineIndex + 1

End Sub
```

## 12.1.2 `_IVScriptEngineEvents::OnVScriptFinished`

```
HRESULT OnVScriptFinished (
    [in] BSTR script_name,
    [in] VS_RESULT result,
    [in] int TAG )
```

Fired when the verification script stops running.

### Parameters

script_name	Name of the verification script
result	Result of the "verification", see <i>IPVerificationScript::RunVerificationScript</i> method, Page 43, for details
TAG	VSE object's tag

### Return values

### Remarks

Make sure that C++ event handlers have `__stdcall` calling convention.

### Example

```
C++:
HRESULT __stdcall CComplTestSink::OnVScriptFinished(
    BSTR script_name,
    VS_RESULT result, int TAG )
{
    USES_CONVERSION;

    TCHAR tmp[220];
    sprintf( tmp, "Script completed, name : %s, result = %d, TAG = %d",
        W2A(script_name),
        result, TAG );

    . . .

    return S_OK;
}
```

### VBA (MS Excel):

```
Public WithEvents VSEEvents As VScriptEngine
. . .

Private Sub VSEEvents_OnVScriptFinished( ByVal script_name As String,
    ByVal result As PEAutomationLib.VS_RESULT,
    ByVal Tag As Long )

    Dim ResString As String
    ResString = "Script name : " & script_name & ", result = " &
        CStr(result) & ", TAG = " & CStr(Tag)

    ThisWorkbook.Sheets("Sheet1").Cells(7, 2) = ResString
End Sub
```

### 12.1.3 `_IVScriptEngineEvents::OnNotifyClient`

```
HRESULT OnNotifyClient(
    [in] int eventId,
    [in] VARIANT eventBody,
    [in] int TAG )
```

Fired when running a verification script, calls the *NotifyClient()* function.

#### Parameters

eventId	Event Id
eventBody	Body of event packed in a VARIANT object
TAG	VSE object's tag

#### Return values

#### Remarks

The information packed in the event body is opaque for VSE – it only packs the information given to *NotifyClient()* function inside of verification script into a VARIANT object and sends it to client applications. See the *PCIe Protocol Analysis Verification Script Engine Manual* for details about the *NotifyClient()* script function.

#### Example

PETracer Verification script:

```
ProcessEvent()
{
    ...
    NotifyClient( 2, [in.Index, in.Level, GetChannelName(), GetEventName(), TimeToText( in.Time
    ] ) );
    ...
}
```

VBA (MS Excel):

```
Public WithEvents VSEEvents As VScriptEngine
...
Private Sub VSEEvents_OnNotifyClient( ByVal eventId As Long,
    ByVal eventBody As Variant,
    ByVal Tag As Long )
    Dim Col As Integer
    Dim Item As Variant

    ThisWorkbook.Sheets("Sheet1").Cells(LineIndex, 1) = eventId

    If IsArray(eventBody) Then
        Col = 3

        For Each Item In eventBody
            ThisWorkbook.Sheets("Sheet1").Cells(LineIndex, Col) = Item
            Col = Col + 1
        Next
    Else
        ThisWorkbook.Sheets("Sheet1").Cells(LineIndex, 2) = eventBody
    End If
```

```
LineIndex = LineIndex + 1  
End Sub
```

## 13 PEAnalyzer Object Events

### 13.1 \_IAnalyzerEvents dispinterface

In order to retrieve the events from a *PEAnalyzer* object, you must implement the *\_IAnalyzerEvents* interface. Since this interface is default source interface for the *PEAnalyzer* object, there is very simple implementation from such languages like Visual Basic, VBA, VBScript, WSH, etc.

Some script engines impose restrictions on handling events from "indirect" automation objects in typeless script languages (when the automation interface to the object is obtained from a call of some method rather than from a creation function – like *CreateObject()* in VBScript). The PCIe Protocol Analysis™ provides a special COM class allowing receiving and handling notifications from the VSE object even in script languages not supporting event handling from "indirect" objects. Please refer to *CATCAalyzerAdapter*, Page 123, for details.

C++ implementation used in the examples below utilizes a sink object by deriving it from *IDispEventImpl*, but not specifying the type library as a template argument. Instead, the type library and default source interface for the object are determined using *AtlGetObjectSourceInterface()*. A *SINK\_ENTRY()* macro is used for each event from each source interface that is to be handled:

```
class CAnalyzerSink : public IDispEventImpl<IDC_SRCOBJ, CAnalyzerSink>
{
BEGIN_SINK_MAP(CAnalyzerSink)
    //Make sure the Event Handlers have __stdcall calling convention
    SINK_ENTRY(IDC_SRCOBJ, 1, OnTraceCreated)
    SINK_ENTRY(IDC_SRCOBJ, 2, OnStatusReport)
END_SINK_MAP()
    . . .
}
```

Then, after you've established a connection with the server, you need to advise as to your implementation of the event interface:

```
hr = CoCreateInstance( CLSID_PEAnalyzer, NULL,
    CLSCTX_SERVER, IID_IPEAnalyzer, (LPVOID *)&m_poPEAnalyzer );

poAnalyzerSink = new CAnalyzerSink();

// Make sure the COM object corresponding to pUnk implements IProvideClassInfo2 or
// IPersist*. Call this method to extract info about source type library if you
// specified only 2 parameters to IDispEventImpl
hr = AtlGetObjectSourceInterface(m_poPEAnalyzer, &poAnalyzerSink->m_libid,
    &poAnalyzerSink->m_iid, &poAnalyzerSink->m_wMajorVerNum,
    &poAnalyzerSink->m_wMinorVerNum);

if ( FAILED(hr) )
    return 1;

// connect the sink and source, m_poPEAnalyzer is the source COM object
hr = poAnalyzerSink->DispEventAdvise(m_poPEAnalyzer, &poAnalyzerSink->m_iid);

if ( FAILED(hr) )
    return 1;
```

### 13.1.1 `_IAnalyzerEvents::OnTraceCreated`

```
HRESULT OnTraceCreated (
    [in] IDispatch* trace )
```

Fired when a trace is created. This event is a result of `IAnalyzer::StartRecording` and `IAnalyzer::StopRecording` method calls (see Pages 10, 12).

#### Parameters

trace                                   Interface pointer to the `PETrace` object

#### Remarks

Make sure the event handlers have `__stdcall` calling convention.

#### Example

VBScript:

```
<OBJECT
    ID = Analyzer
    CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>
<P ALIGN=LEFT ID=StatusText></P>

<SCRIPT LANGUAGE="VBScript">
<!--
Dim CurrentTrace
Sub Analyzer_OnTraceCreated(ByRef Trace)
    On Error Resume Next
    Set CurrentTrace = Trace
    If Err.Number <> 0 Then
        MsgBox Err.Number & ":" & Err.Description
    End If
    StatusText.innerText = "Trace '" & CurrentTrace.GetName & "' created"
End Sub
-->
</SCRIPT>
```

C++:

```
HRESULT __stdcall OnTraceCreated( IDispatch* trace )
{
    IPETrace* pe_trace;
    HRESULT hr;
    hr = trace->QueryInterface( IID_IPETrace, (void*)&pe_trace );

    if (FAILED(hr))
    {
        _com_error er(hr);
        if (er.Description().length() > 0)
            ::MessageBox( NULL, er.Description(), _T("PETracer client"), MB_OK );
        else
            ::MessageBox( NULL, er.ErrorMessage(), _T("PETracer client"), MB_OK );
        return hr;
    }

    . . .

    return hr;
}
```

### 13.1.2 `_IAnalyzerEvents::OnStatusReport`

```
HRESULT OnStatusReport (
    [in] short subsystem,
    [in] short state,
    [in] long percent_done )
```

Fired when there is a change in the analyzer's state, or there is a change in progress (percent\_done) of the analyzer's state.

#### Parameters

subsystem	Subsystem sending event has the following values: RECORDING_PROGRESS_REPORT ( 1 ) recording subsystem GENERATION_PROGRESS_REPORT ( 2 ) generation subsystem
state	Current analyzer state; has the following values: If the <i>subsystem</i> is RECORDING_PROGRESS_REPORT: ANALYZERSTATE_IDLE ( -1 ) Idle ANALYZERSTATE_WAITING_TRIGGER ( 0 ) Recording in progress, analyzer is waiting for trigger ANALYZERSTATE_RECORDING_TRIGGERED ( 1 ) Recording in progress, analyzer triggered ANALYZERSTATE_UPLOADING_DATA ( 2 ) Uploading in progress ANALYZERSTATE_SAVING_DATA ( 3 ) Saving data in progress ANALYZERSTATE_PREPARE_UPLOADING ( 4 ) Analyzer is preparing data for upload ANALYZERSTATE_PREPARE_RECORDING ( 5 ) Analyzer is preparing for recording If the <i>subsystem</i> is GENERATION_PROGRESS_REPORT: ANALYZERSTATE_GEN_IDLE ( 400 ) Generator is idle ANALYZERSTATE_GEN_DOWNLOADING ( 401 ) Generator is downloading object code ANALYZERSTATE_GEN_GENERATING ( 402 ) Generator is working ANALYZERSTATE_GEN_PAUSED ( 403 ) Generator is paused
percent_done	Shows the progress of currently performing operation If <i>subsystem</i> is RECORDING_PROGRESS_REPORT: <ul style="list-style-type: none"> <li>• When analyzer state is ANALYZERSTATE_IDLE, this parameter is not applicable.</li> <li>• When analyzer state is ANALYZERSTATE_WAITING_TRIGGER or ANALYZERSTATE_RECORDING_TRIGGERED, this parameter shows analyzer memory utilization</li> <li>• When Analyzer state is ANALYZERSTATE_UPLOADING_DATA, this parameter shows the percent of data uploaded.</li> <li>• When Analyzer state is ANALYZERSTATE_SAVING_DATA, this parameter shows the percent of data saved.</li> </ul>

If *subsystem* is GENERATION\_PROGRESS\_REPORT:

- Represent current position of the script execution

## Return values

## Remarks

Make sure the event handlers have `__stdcall` calling convention.

## Example

VBScript:

```
<OBJECT
  ID = Analyzer
  CLASSID = "clsid: 297CD804-08F5-4A4F-B3BA-779B2654B27C "
>
</OBJECT>
<P ALIGN=LEFT ID=StatusText></P>

<SCRIPT LANGUAGE="VBScript">
<!--
Function GetRecordingStatus(ByVal State, ByVal Percent)
  Select Case State
    Case -1: GetRecordingStatus = "Idle"
    Case 0: GetRecordingStatus = "Recording - Waiting for trigger"
    Case 1: GetRecordingStatus = "Recording - Triggered"
    Case 2: GetRecordingStatus = "Uploading"
    Case 3: GetRecordingStatus = "Saving Data"
    Case 4: GetRecordingStatus = "Preparing Data for Upload"
    Case 5: GetRecordingStatus = "Preparing Data for Recording"
    Case Else: GetRecordingStatus = "Invalid recording status"
  End Select
  GetRecordingStatus = GetRecordingStatus & ", " & Percent & "% done"
End Function

Dim RecordingStatus
Sub Analyzer_OnStatusReport(ByVal System, ByVal State, ByVal Percent)
  Select Case System
    Case 1 RecordingStatus = GetRecordingStatus( State, Percent )
  End Select
End Sub
-->
</SCRIPT>
```

```

C++:
#define RECORDING_PROGRESS_REPORT          ( 1 )
#define ANALYZERSTATE_IDLE                 ( -1 )
#define ANALYZERSTATE_WAITING_TRIGGER     ( 0 )
#define ANALYZERSTATE_RECORDING_TRIGGERED ( 1 )
#define ANALYZERSTATE_UPLOADING_DATA     ( 2 )
#define ANALYZERSTATE_SAVING_DATA        ( 3 )
#define ANALYZERSTATE_PREPARE_UPLOADING  ( 4 )
#define ANALYZERSTATE_PREPARE_RECORDING  ( 5 )

HRESULT __stdcall OnStatusReport( short subsystem, short state, long percent_done )
{
    switch ( subsystem )
    {
        case RECORDING_PROGRESS_REPORT:
            UpdateRecStatus( state, percent_done );
            break;
    }
    TCHAR buf[1024];
    _stprintf( buf, _T("%s"), m_RecordingStatus );
    ::SetWindowText( m_hwndStatus, buf );

    return S_OK;
}

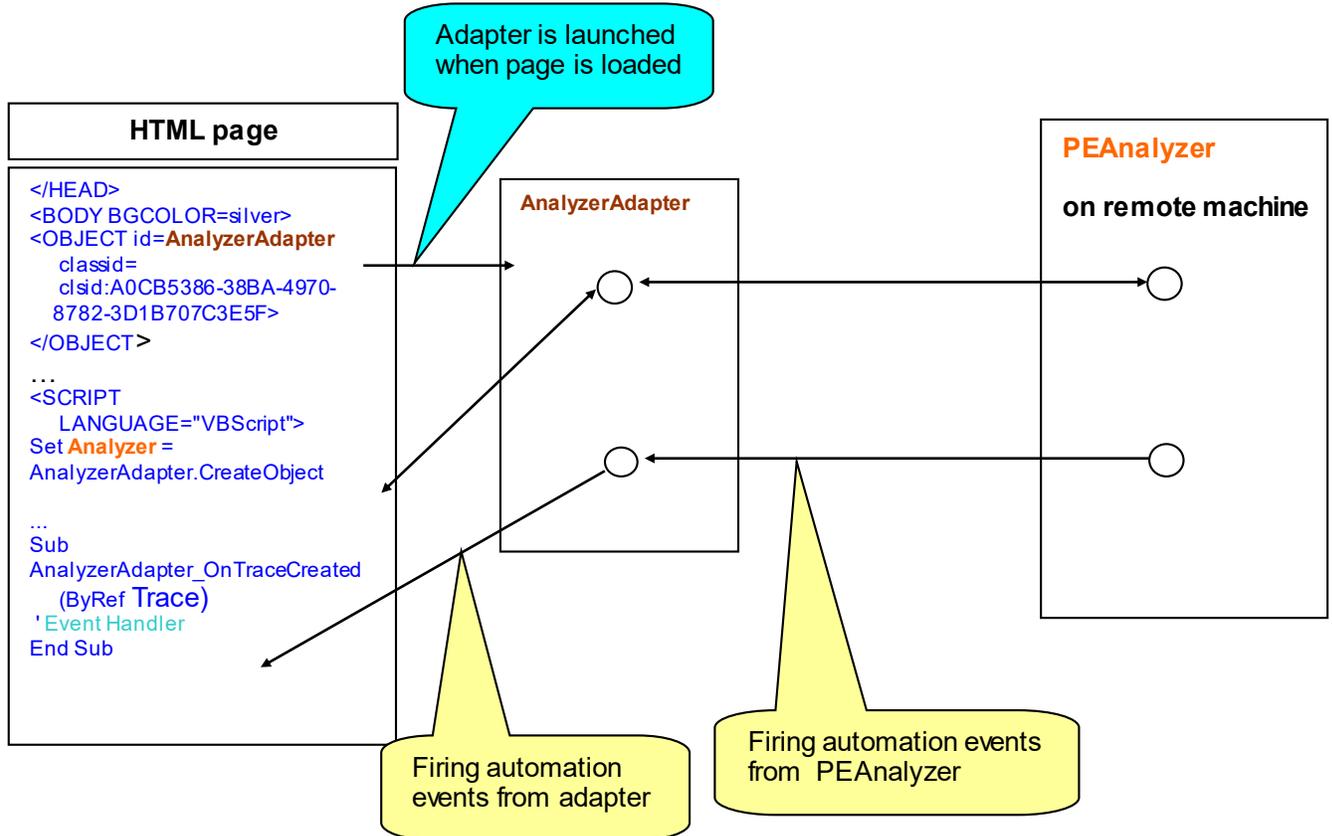
void UpdateRecStatus( short state, long percent_done )
{
    TCHAR status_buf[64];
    switch ( state )
    {
        case ANALYZERSTATE_IDLE:
            _tcscpy( status_buf, _T("Idle") );
            break;
        case ANALYZERSTATE_WAITING_TRIGGER:
            _tcscpy( status_buf, _T("Recording - Waiting for trigger") );
            break;
        case ANALYZERSTATE_RECORDING_TRIGGERED:
            _tcscpy( status_buf, _T("Recording - Triggered") );
            break;
        case ANALYZERSTATE_UPLOADING_DATA:
            _tcscpy( status_buf, _T("Uploading") );
            break;
        case ANALYZERSTATE_SAVING_DATA:
            _tcscpy( status_buf, _T("Saving data") );
            break;
        case ANALYZERSTATE_PREPARE_UPLOADING:
            _tcscpy( status_buf, _T("Preparing Data for Upload") );
            break;
        case ANALYZERSTATE_PREPARE_RECORDING:
            _tcscpy( status_buf, _T("Preparing Data for Recording") );
            break;
        default:
            _tcscpy( status_buf, _T("Unknown") );
            break;
    }
    _stprintf( m_RecordingStatus, _T("%s, done %ld%%"), status_buf, percent_done );
}

```

## 14 CATCAnalyzerAdapter

*CATCAnalyzerAdapter* is an automation server that allows the launching and accessing of Teledyne LeCroy analyzer automation servers. If all necessary DCOM settings and permissions are set on the remote server, the server can be run remotely over an IP network. The examples below shows how the *CATCAnalyzerAdapter* is used as an intermediary between an HTML page and the PCIe Protocol Analysis analyzer server.

The following diagram illustrates how this functionality works:



The Class ID and App ID for *PEAnalyzer* object are the following.

**Class ID:** A0CB5386-38BA-4970-8782-3D1B707C3E5F  
**App ID:** CATC.AnalyzerAdapter

**Primary interface:** *IAnalyzerAdapter*.

## 14.1 IAnalyzerAdapter Interface

### 14.1.1 IAnalyzerAdapter::CreateObject

```
HRESULT CreateObject (
    [in] BSTR class_id,
    [in, optional] BSTR host_name,
    [out, retval] IDispatch** ppNew Obj )
```

This method instantiates the LeCroy analyzer object on a local or remote machine and attaches it to the adapter.

#### Parameters

class_id	String representation of <i>classid</i> or <i>ProgId</i> ( <i>clsid:297CD804-08F5-4A4F-B3BA-779B2654B27C</i> or <i>CATC.PETracer</i> for <i>PEAnalyzer</i> object)
host_name	Network name of the remote server where the analyzer object should be instantiated. Empty value means local host.
ppNewObj	Pointer to the created remote object, NULL if the object has not been instantiated or accessed.

#### Return values

#### Remarks

Only Teledyne LeCroy analyzer COM servers can be instantiated through this method. The method *Detach*, below, should be called when the work with the remote object is completed.

**NOTE:** The pointer returned in *ppNewObj* should be released separately.

#### Example

```
VBScript:
</HEAD>
<OBJECT id=AnalyzerAdapter
    classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
...
<input type="button" value="Connect" name="BtnConnect">
<INPUT NAME="RemoteServer">
<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnConnect_onclick
    On Error Resume Next

    Set Analyzer = AnalyzerAdapter.CreateObject("CATC.PETracer", RemoteServer.value )

    if Not Analyzer Is Nothing Then
        window.status = "PETracer connected"
    else
        msg = "Unable to connect to PETracer"
        MsgBox msg, vbCritical
        window.status = msg
    End If
End Sub
-->
</SCRIPT>
```

WSH:

```
' Create CATC analyzer adapter first..
Set AnalyzerAdapter = WScript.CreateObject("CATC.AnalyzerAdapter", "Analyzer_")

RemoteServer = "EVEREST"
Set Analyzer = AnalyzerAdapter.CreateObject("CATC.PETracer", RemoteServer)

Analyzer.StartRecording ( Analyzer.ApplicationFolder & "my.rec" )
...
```

## 14.1.2 IAnalyzerAdapter::Attach

```
HRESULT Attach(
    [in] IDispatch* pObj )
```

This method attaches the LeCroy analyzer object to the adapter.

### Parameters

**pObj**                      Pointer to the LeCroy analyzer object to be attached.

### Return values

### Remarks

Only LeCroy analyzer COM servers can be attached to the adapter. If some other analyzer object were previously attached to the adapter, it is detached by this call. When the analyzer object gets attached to the adapter, a client application using the adapter becomes able to handle automation events fired by the remote analyzer object through the adapter.

### Example

```
VBScript:
</HEAD>
<OBJECT id=AnalyzerAdapter
    classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
...
<input type="button" value="Connect" name="BtnConnect">

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnConnect_onclick
    On Error Resume Next

    Set Analyzer = CreateObject("CATC.PETracer" ) 'VBScript function    creates object
    locally

    if Not Analyzer Is Nothing Then
        AnalyzerAdapter.Attach Analyzer ' attach analyzer to the adapter

        window.status = "PETracer connected"
    else
        msg = "Unable to connect to PETracer"
        MsgBox msg, vbCritical
        window.status = msg
    End If
End Sub

-->
</SCRIPT>

WSH:
' Create CATC analyzer adapter first..
Set AnalyzerAdapter = WScript.CreateObject("CATC.AnalyzerAdapter", "Analyzer_")

'VBScript functioncreates object locally
Set Adapter = WScript.CreateObject("CATC.AnalyzerAdapter")

AnalyzerAdapter.Attach Analyzer ' Attach analyzer object to the adapter
Analyzer.StartRecording ( Analyzer.ApplicationFolder & "my.rec" )
...
```

### 14.1.3 IAnalyzerAdapter::Detach

```
HRESULT Detach( )
```

This method detaches the LeCroy analyzer object from the adapter.

#### Parameters

#### Return values

#### Remarks

This method detaches an analyzer object from the adapter. This method doesn't guarantee that all resources associated with the detached object is freed. All existing pointers to that object should be released to destroy the remote object.

#### Example

VBScript:

```
</HEAD>
<OBJECT id=AnalyzerAdapter
    classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
...
<input type="button" value="Connect" name="BtnConnect">
<input type="button" value="Disconnect" name="BtnDisconnect">
<INPUT NAME="RemoteServer">

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnConnect_onclick
    On Error Resume Next

    Set Analyzer = AnalyzerAdapter.CreateObject("CATC.PETracer", RemoteServer.value )

    if Not Analyzer Is Nothing Then
        window.status = "PETracer connected"
    else
        msg = "Unable to connect to PETracer"
        MsgBox msg, vbCritical
        window.status = msg
    End If
End Sub

Sub BtnDisconnect_OnClick
    AnalyzerAdapter.Detach ' Detach the analyzer object from adapter
    Set Analyzer = Nothing ' Release the pointer to the analyzer returned by
                           CreateObject()

    window.status = "PETracer disconnected"
End Sub
-->
</SCRIPT>
```

WSH:

```
' Create CATC analyzer adapter first..
Set AnalyzerAdapter = WScript.CreateObject("CATC.AnalyzerAdapter", "Analyzer_")

RemoteServer = "EVEREST"
Set Analyzer = AnalyzerAdapter.CreateObject("CATC.PETracer", RemoteServer)

Analyzer.StartRecording ( Analyzer.ApplicationFolder & "my.rec" )
...
AnalyzerAdapter.Detach    ' - Disconnect the remote analyzer from the adapter
Set Analyzer = Nothing    ' - Release the analyzer ...

'Release the adapter ...
Set AnalyzerAdapter = Nothing
```

### 14.1.4 IAnalyzerAdapter::IsValidObject

```
HRESULT IsValidObject(
    [in] IDispatch *pObj,
    [out,retval] VARIANT_BOOL* pVal )
```

This method helps to determine whether some automation object can be attached to the adapter.

#### Parameters

pObj	Pointer to the object validated
pVal	Pointer to the variable receiving result. TRUE if the validated object can be attached, FALSE otherwise

#### Return values

#### Remarks

Only LeCroy analyzer COM servers can be attached to the adapter.

#### Example

VBScript:

```
</HEAD>
<OBJECT id=AnalyzerAdapter
    classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
...
<input type="button" value="Connect" name="BtnConnect">
<input type="button" value="Disconnect" name="BtnDisconnect">
<INPUT NAME="RemoteServer">

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnConnect_onclick

    'Launch MS Excel instead of PCIe Protocol Analysis !!!
    Set Analyzer = CreateObject("Excel.Application")
    Analyzer.Visible = True

    If Not AnalyzerAdapter.IsValidObject( Analyzer ) Then
        MsgBox "The object cannot be attached", vbCritical
        Set Analyzer = Nothing
        Exit Sub
    End If
End Sub

-->
</SCRIPT>
```

## 15 Teledyne LeCroy Internal Interfaces

The following interfaces are for Teledyne LeCroy use only and should not be used by any other applications:

IPSGViewSyncClient  
IPSGViewSyncContext  
IPSGViewSyncTrace  
IPECalibration.

## 16 Appendix A

### How to Contact Teledyne LeCroy

Send e-mail...	<a href="mailto:psgsupport@teledynelecroy.com">psgsupport@teledynelecroy.com</a>
Contact support...	<a href="http://teledynelecroy.com/support/contact">teledynelecroy.com/support/contact</a>
Visit Teledyne LeCroy's web site...	<a href="http://teledynelecroy.com">teledynelecroy.com</a>
Tell Teledyne LeCroy...	Report a problem to Teledyne LeCroy Support via e-mail by selecting <b>Help &gt; Tell Teledyne LeCroy</b> from the application toolbar. This requires that an e-mail client be installed and configured on the host machine.