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

# Automation API

# for

# LeCroy PE*Tracer*/PE*Trainer*™

# Reference Manual

**Manual Version 1.7**

**For PE*Tracer* Software Version 5.00**

June 2006

# Document Disclaimer

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

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

# Trademarks and Servicemarks

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

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

All other trademarks are property of their respective companies.

# Copyright

# Version

This is version 1.7 of the PE*Tracer*/*Trainer* Automation API Reference Manual. This manual applies to PE*Tracer* software version 5.00 and higher.

# Table of Contents

# 1 Introduction

LeCroy's PE*Tracer*™ software provides a rich, functional COM/Automation API to the most important functionalities of the LeCroy PE*Tracer* Protocol Analyzer and LeCroy PE*Trainer*™ 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 LeCroy PE*Tracer* 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 PE*Tracer* 4.00

## 1.2 Support Resources

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

## 1.3 Setting Up Automation for Local Use

If you intend to run Automation on the PE*Tracer*/PE*Trainer* Host Controller (i.e., the PC attached to the PE*Tracer*/PE*Trainer*), 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 PE*Tracer* 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

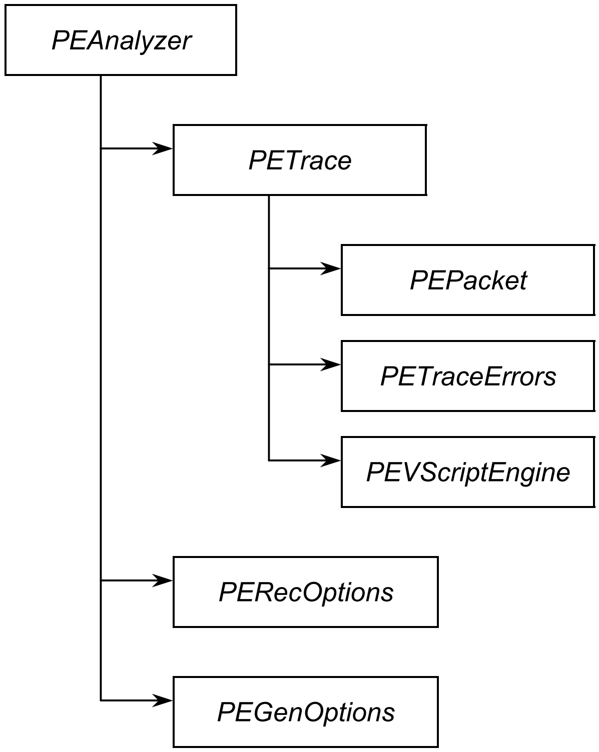
If you would like to access PE*Tracer* remotely over a network, you should install the PE*Tracer* application on both server and client machine and accept enabling remote access option during the installation. You can also perform a manual DCOM configuration.

# 2 PE*Tracer* Object Model

LeCroy's PE*Tracer*™ 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 LeCroy PE*Tracer* 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.

| **Objects** | **Interfaces** | **Description** |
|---|---|---|
| *PEAnalyzer* | *IAnalyzer*<br>*IPEAnalyzer*<br>*IPEAnalyzer2**<br>*_IAnalyzerEvents* | Represents the PE*Tracer* application |
| *PETrace* | *ITrace*<br>*IPETrace**<br>*IPEVerificationScript** | Represents recorded trace |
| *PERecOptions* | *IRecOptions*<br>*IPERecOptions*<br>*IPERecOptions2** | Represents recording options |
| *PEGenOptions* | *IGenOptions*<br>*IPEGenOptions*<br>*IPEGenOptions2** | Represents generation options |
| *PEPacket* | *IPacket*<br>*IPEPacket** | Represents single packet of the recorded trace |
| *PETraceErrors* | *IAnalyzerErrors** | 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.

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

# 3 PEAnalyzer Object

The *PEAnalyzer* object is a top-level object of PE*Tracer*™ 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 |
|---|---|
| *IAnalyzer* | Facilitates recording and traffic generation, opens trace files, and retrieves recording options, |
| *IPEAnalyzer* | Extends the *IAnalyzer* interface:<br>Adds advanced generator functionality, retrieves generation options. |
| *IPEAnalyzer2* | Extends the *IPEAnalyzer* interface:<br>Adds hardware information and control methods. |
| *_IAnalyzerEvents* | Events from *PEAnalyzer* object |

The *IPEAnalyzer2* 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_IPEAnalyzer,
                (LPVOID *)&poPEAnalyzer ) )
               return;
```

# 3.1 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*

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

### 3.1.1 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; *EAnalyzerVersionType* 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];
       _stprintf(buffer, _T("Software version:%X.%X"), HIBYTE(sw_version), LOBYTE(sw_version));
```

## 3.1.2 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 *PETrace* 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;
```

## 3.1.3 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;
}
```

### 3.1.4 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;
        }
```

### 3.1.5 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 PE*Tracer* application. You can create this file when you select "*Setup -> Recording Options…*" from the PE*Tracer* 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;
        }
```

## 3.1.6 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 *_IAnalizerEvents* 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;
        }
```

### 3.1.7 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 *PETrace* 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 PE*Tracer* application. You can create this file when you select “*Setup -> Recording Options…*” from the PE*Tracer* 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();
```

### 3.1.8 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 PE*Tracer* application. You can create this file when you select “*Setup -> Display Options…*” from the PE*Tracer* application menu, change the settings in the “Display Options” dialog box, and then select the “*Save…*” button.

**Example**

See *ITrace::ApplyDisplayOptions*, Page 27.

### 3.1.9 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;
```

# 3.2 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*
*GetLinkStatus*

**Note:** All methods implemented by the *IPEAnalyzer* interface are also implemented by the *IPEAnalyzer2* interface (see Page 20).

### 3.2.1 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;
```

## 3.2.2 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;
        }
```

### 3.2.3 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 );
```

# 3.3 IPEAnalyzer2 interface

The *IPEAnalyzer2* interface is a primary 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*

## 3.3.1 IPEAnalyzer2::GetHardwareInfo

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

Returns information about the hardware (PE*Tracer*/PE*Trainer*™) connected.

**Parameters**

`type` Hardware type being queried; the *EHardwareType* enumerator has the following values:

`HARDWARETYPE_PETRACER ( 0 ) –` PE*Tracer*
`HARDWARETYPE_PETRAINER ( 1 ) -` PE*Trainer*

`info` The following values can be returned

When `type` is `HARDWARETYPE_PETRACER:`
1 – PE*Tracer* ML
2 – PE*Tracer* ML (2 boxes)
3 – PE*Tracer* EML

When `type` is `HARDWARETYPE_PETRACER:`
1 – PE*Trainer* ML
2 – PE*Trainer* EML

**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;
       }
```

### 3.3.2 IPEAnalyzer2::ResetHardware

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

Resets the hardware specified.

**Parameters**

`type` Hardware type to reset; the *EHardwareType* enumerator has the following values:

`HARDWARETYPE_PETRACER ( 0 )` – PE*Tracer*
`HARDWARETYPE_PETRAINER ( 1 )` - PE*Trainer*

`reset_type` Type of the reset; the *EResetType* enumerator has the following values:

`RESETTYPE_LINK ( 0 )` – link reset

**Remarks**

**Example**

C++:

```
IPEAnalyzer2* poAnalyzer;

. . .


try
{
      poAnalyzer->ResetHardware( HARDWARETYPE_PETRAINER, RESETTYPE_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;
}
```

# 4 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 7)
- Using *IAnalyzer::MakeRecording* method (see Page 13)
- Handling *_IAnalyzerEvents::OnTraceCreated* event (see Page 120)

The *PETrace* object supports the following interfaces:

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

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

# 4.1 ITrace interface

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

It implements the following methods:

*GetName*<br>
*ApplyDisplayOptions*<br>
*Save*<br>
*ExportToText*<br>
*Close*<br>
*ReportFileInfo*<br>
*ReportErrorSummary*<br>
*GetPacket*<br>
*GetPacketsCount*<br>
*GetTriggerPacketNum*<br>
*AnalyzerErrors*

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

### 4.1.1 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.PETracer” )
       CurrentDir = Left( WScript.ScriptFullName, InstrRev( WScript.ScriptFullName, “\” ) )
       Set Trace = Analyzer.MakeRecording( CurrentDir & "Input\test_ro.rec" )
       MsgBox "Trace name " & Trace.GetName

C++:
       IPETrace* 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("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 );

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

### 4.1.2 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 PE*Tracer*™ application. You can create this file when you select “*Setup -> Display Options…*” from the PE*Tracer* 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;
}
```

## 4.1.3 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` | *beginning packet number* 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` | *ending packet number* 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;
       }
```

## 4.1.4 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 | *beginning packet number* 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 | *ending packet number* 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;
       }
```

### 4.1.5 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**

## 4.1.6 ITrace::ReportFileInfo

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

Saves trace information into a specified HTML file.

### Parameters

`file_name` String providing the full pathname to a file where the trace information report is stored

### Return values

`ANALYZERCOMERROR_UNABLESAVE` 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;
        }
```

### 4.1.7 ITrace::ReportErrorSummary

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

Saves trace error summary information into the specified text file.

**Parameters**

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

**Return values**

| | |
|---|---|
| `ANALYZERCOMERROR_UNABLESAVE` | 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;
       }
```

### 4.1.8 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 );
        }
```

## 4.1.9 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 );
```

## 4.1.10 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 occured |

### 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.

## 4.1.11 ITrace::AnalyzerErrors

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

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

**Parameters**

`error_type` Type of error collection you want to retrieve; the following values are valid:

| | |
|---|---|
| `0x00000001` | - Bad ECRCs |
| `0x00000002` | - Bad LCRCs |
| `0x00000004` | - Bad Packet length |
| `0x00000008` | - Alignment Error |
| `0x00000010` | - Invalid 10b Code |
| `0x00000020` | – Running Disparity Error |
| `0x00000040` | – End of Bad Packet |
| `0x00000080` | – Delimiter Error |
| `0x00000100` | – TS Data Error |
| `0x00000200` | – Ordered Set Format Error |
| `0x00000400` | – Idle Error |
| `0x00000800` | – Skip Late |
| `0x00001000` | – Skew Error |

`analyzer_errors` Address of a pointer to the *PETraceErrors* object interface

**Return values**

`ANALYZERCOMERROR_INVALIDERROR` Invalid error type specified

**Remarks**

*PETraceErrors* 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();
```

# 4.2 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*

## 4.2.1 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 *PEPacket* 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();
```

# 4.3 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 PE*Tracer* application, its *Verification Script Engine*.

Please refer to the *PETracer Manual*, the *PETracer Verification Script Engine Manual*, and the *PETracer 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.

## 4.3.1 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; *VS_RESULT* is an enumeration type that can have 5 possible meanings:

| | | | |
|---|---|---|---|
| `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, PE*Tracer*’s server is going to search for the named script among the scripts loaded from the *\Scripts\VFScripts* folder under PE*Tracer* 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 *PETracer Verification Script Engine Manual* for more details.

## Example

```
C++:

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

        IPEVerificationScript* vscript = NULL;

        if ( SUCCEEDED ( trace->QueryInterface( IID_IPEVerificationScript, (void**)&vscript ) )
        {
                try
                {

                    VS_RESULT result = vscript ->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" )
```

### 4.3.2 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 *PEVScriptEngine* 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_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();
        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" )
```

# 5 PERecOptions Object

The *PERecOptions* object represents the options for the PE*Tracer*™ (x1, ML, EML) 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** |
|---|---|
| *IRecOptions* | 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 |
| *IPERecOptions* | Identical to *IRecOptions* interface |
| *IPERecOptions2* | Extends the *IPERecOptions* 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.

# 5.1 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.

## 5.1.1 IRecOptions::Load

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

Loads recording options from the specified file.

**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++:

### 5.1.2 IRecOptions::Save

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

Saves recording options into the specified file.

**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++:
```

## 5.1.3 IRecOptions::SetRecMode

```
HRESULT SetRecMode (
      [in] ERecModes rec_mode )
```

Sets the recording mode.

**Parameters**

`rec_mode` Enumerated value providing the mode to set; *ERecModes* 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++:

### 5.1.4 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 is 1MB for PE*Tracer* x1, 16MB for PE*Tracer* ML, and 32MB for PE*Tracer* EML.

**Example**

WSH:

```
Set Analyzer = WScript.CreateObject( “CATC.PETracer” )
Set RecOptions = Analyzer.GetRecordingOptions( )
RecOptions.SetBufferSize 2*1024*1024  ' 2Mb
```

C++:

### 5.1.5 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++:

### 5.1.6 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,<br>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++:
```

## 5.1.7 IRecOptions::SetSaveExternalSignals

```
HRESULT SetSaveExternalSignals (
      [in] BOOL save )
```

Sets a flag to save external signals.

**Parameters**

| | |
|---|---|
| `save` | TRUE – save external signals,<br>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++:

### 5.1.8 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++:
```

### 5.1.9 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++:
```

# 5.2 IPERecOptions interface

This interface is identical to the *IRecOptions* interface (see Page 48).

# 5.3 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*

## 5.3.1 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; *ETargetAnalyzer* has the following values:

```
TARGETANALZYER_X1  (0) - PETracer x1
TARGETANALZYER_ML  (1) - PETracer ML
TARGETANALZYER_ML2 (2) - PETracer ML (2 boxes)
TARGETANALZYER_EML (3) - PETracer EML
```

## 5.3.2 IPERecOptions2::SetLinkWidth

```
HRESULT SetLinkWidth (
      [in] int link_width )
```

Sets the link width.

**Parameters**

`link_width` Link width to set; Allowed values are

```
PETracer x1       - 1
PETracer ML       - 1, 2, 4, 8
PETracer EML      - 1, 2, 4, 8, 16
```

## 5.3.3 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 PE*Tracer* hardware uses Base Spec 1.0 compatibility mode |

**Remarks**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

### 5.3.4 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 |

### 5.3.5 IPERecOptions2::SetDisableDescrambling

```
HRESULT SetDisableDescrambling (
      [in] BOOL disable_descrambling )
```

Disables/enables descrambling of incoming traffic.

**Parameters**

`disable_descrambling` When TRUE, the descrambling is disabled

**Remarks**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

## 5.3.6 IPERecOptions2::SetDisableDeskew

```
HRESULT SetDisableDeskew(
      [in] BOOL disable_deskew )
```

Disables/enables deskew of incoming traffic.

**Parameters:**

| | |
|---|---|
| `disable_deskew` | When TRUE, the deskew is disabled |

**Remarks**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

### 5.3.7 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**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

## 5.3.8 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; *EDirection* has the following values:

```
DIRECTION_UPSTREAM   (0) - upstream traffic
DIRECTION_DOWNSTREAM (1) - downstream traffic
```

`inhibit` Specifies whether to inhibit traffic specified in the *direction* parameter

**Remarks**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

## 5.3.9 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;
*EDirection* 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**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1.

### 5.3.10 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; *EDirection* 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**

Implemented for PE*Tracer* ML and PE*Tracer* EML. Not implemented for PE*Tracer* x1. This call fails if automatic polarity detection is enabled (see *IPERecOptions2::SetAutoConfigPolarity*, Page 64)

# 6 PEGenOptions Object

The *PEGenOptions* object represents the options for the PE*Trainer*™ (ML and EML) 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 17)

The *PEGenOptions* object supports the following interfaces:

| Interfaces | Description |
|---|---|
| *IGenOptions* | Allows you to load/save recording options from/to the file and reset generation options |
| *IPEGenOptions* | Identical to *IGenOptions* interface |
| *IPEGenOptions2* | Extends the *IPEGenOptions* 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.

# 6.1 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.

### 6.1.1 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++:

## 6.1.2 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++:
```

### 6.1.3 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++:

# 6.2 IPEGenOptions interface

This interface is identical to the *IGenOptions* interface (see Page 69).

# 6.3 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*<br>
*SetWorkAsRoot*<br>
*SetLinkWidth*<br>
*SetBase10Spec*<br>
*SetExternalRefClock*<br>
*SetDisableDescrambling*<br>
*SetDisableScrambling*<br>
*SetAutoConfig*<br>
*SetReverseLanes*<br>
*SetInvertPolarity*<br>
*SetSkew*

### 6.3.1 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; *ETargetGenerator* has the following values:

```
TARGETGENERATOR_ML      (0) – PETrainer ML
TARGETGENERATOR_EML     (1) – PETrainer EML
```

**Return values**

**Remarks**

**Example**

WSH:

```
Set Analyzer = WScript.CreateObject( “CATC.PETracer” )
Set GenOptions = Analyzer.GetGenerationOptions( )
GenOptions.SetTargetGenerator( 1 )
```

C++:

### 6.3.2 IPEGenOptions2::SetWorkAsRoot

```
HRESULT SetWorkAsRoot
      ( [in] BOOL root )
```

**Parameters**

`root` If TRUE, then the PE*Trainer* 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++:

### 6.3.3 IPEGenOptions2::SetLinkWidth

```
HRESULT SetLinkWidth (
      [in] int link_width )
```

Sets the link width.

**Parameters:**

| | |
|---|---|
| `link_width` | Link width to set; allowed values are<br>PE*Trainer* ML - 1, 4, 8<br>PE*Trainer* EML - 1, 4, 8, 16 |

**Return values**

**Remarks**

**Example**

WSH:

```
Set Analyzer = WScript.CreateObject( “CATC.PETracer” )
Set GenOptions = Analyzer. GetGenerationOptions
GenOptions.SetLinkWidth( 1 )
```

C++:

### 6.3.4 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 PE*Trainer* 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++:

### 6.3.5 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**

Implemented for PE*Trainer* ML. Not implemented for PE*Trainer* EML.

## 6.3.6 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++:

### 6.3.7 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++:

### 6.3.8 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:

- Link width (PE*Trainer* ML only)
- Scrambling of outgoing traffic
- Descrambling of incoming traffic
- Lane reversal
- Polarity inversion of incoming traffic

**Return values**

**Remarks**

**Example**

WSH:

```
Set Analyzer = WScript.CreateObject( “CATC.PETracer” )
Set GenOptions = Analyzer.GetGenerationOptions( )
GenOptions.SetAutoConfig( 0 )
```

C++:

## 6.3.9 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;
*EDirection* 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++:

## 6.3.10 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;
*EDirection* 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++:

## 6.3.11 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++:
```

# 7 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** |
|---|---|
| *IPacket* | Allows retrieval of the packet's timestamp |
| *IPEPacket* | Extends the *IPacket* interface |

The *IPEPacket* interface is a primary interface for the *PEPacket* object.

# 7.1 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).

## 7.1.1 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++:

# 7.2 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*

### 7.2.1 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 *EPacketFormat* enumerator has the following values:

```
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 );
        }
```

## 7.2.2 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++:

### 7.2.3 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++:

### 7.2.4 IPEPacket::GetLFSR

```
HRESULT GetLFSR (
      [out, retval] long* lfsr )
```

Returns Linear Feedback Shift Register (LFSR) value before the packet:

**Parameters**

| | |
|---|---|
| `lfsr` | LFSR value before 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 "LFSR: " & Packet.GetLFSR
```

C++:

### 7.2.5 IPEPacket::GetDirection

```
HRESULT GetDirection (
      [out, retval] long* direction )
```

Returns direction (upstream/downstream) of this packet.

**Parameters**

| | |
|---|---|
| `direction` | 0 – upstream packet<br>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++:

### 7.2.6 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 *ITrace::AnalyzerErrors*, Page 39, for error id values

`number_of_errors` Total number of errors in this packet

**Return values**

**Remarks**

**Example**

```
WSH:
```

```
C++:
```

# 8 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 41).

The *IAnalyzerErrors* interface is a primary interface for the *PETraceErrors* object.

## 8.1 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*

### 8.1.1 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 |

## 8.1.2 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 occured
        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++:

```
        IPETrace* 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("PETracer client"), MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(),_T("PETracer 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("PETracer client"), MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(),_T("PETracer client"), MB_OK );
                return 1;
        }

        analyser_errors->Release();

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

# 9 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 |
|---|---|
| *IVScriptEngine* | Provides advanced control over the verification script and allows you to execute the script asynchronously |
| *_IAnalyzerEvents* | Events from *PEVScriptEngine* 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 PE*Tracer*™ application, its *Verification Script Engine*.

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

# 9.1 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*

### 9.1.1 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 PE*Tracer* server is going to search for the named script among the scripts loaded from the \Scripts\VFScripts folder under PE*Tracer* 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 );
         }

        . . .
```

### 9.1.2 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_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 );

 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 );
 }

 . . .
```

### 9.1.3 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.

## 9.1.4 IVScriptEngine::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 PE*Tracer* server is going to search for the named script among the scripts loaded from the \Scripts\VFScripts folder under PE*Tracer* 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_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();
        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 );
        }

        . . .
```

### 9.1.5 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_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();

        // 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.
        . . .
```

### 9.1.6 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();
        . . .
```

### 9.1.7 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 *PETracer 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 *PETracer File Based Decoding Manual* for more details about list objects)

**Parameters**

`var_name` String providing the name of the global variable or constant used in the verification script running

`var_value` Address of a VARIANT variable where the result is kept

**Return values**

`E_PENDING` 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

               MyIntVar = VSEngine.GetScriptVar( "MyIntVar" ) ' Let's suppose that MyIntVar
contains an integer
               MyStrVar = VSEngine.GetScriptVar( "MyStrVar" ) ' Let's suppose that MyStrVar
contains a string

               MsgBox " MyIntVar = " & CStr(MyIntVar) & ", MyStrVar = " & MyStrVar
```

### 9.1.8 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 *PETracer 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 *PETracer 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
```

# 10 PEVScriptEngine Object Events

## 10.1_IVScriptEngineEvents interface

In order to retrieve the event notifications from PE*Tracer*™ 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 PE*Tracer* 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
```

### 10.1.1 _ 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 *PETracer Verification Script Engine Manual* for details on the *ReportText* function).

**Parameters**

| | |
|---|---|
| `newLine` | New portion of text reported by the verification script |
| `TAG` | 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
```

## 10.1.2 _ 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 *IPEVerificationScript::RunVerificationScript* 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
```

### 10.1.3 _ 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 *PETracer 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
```

# 11 PEAnalyzer Object Events

## 11.1_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 PE*Tracer*™ 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 *CATCAnalyzerAdapter*, 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;
```

### 11.1.1 _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;
        }
```

### 11.1.2 _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 *subsystem* 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

If the *subsystem* is `GENERATION_PROGRESS_REPORT`:
`ANALYZERSTATE_GEN_IDLE ( 400 )` Generator is idle
`ANALYZERSTATE_GEN_DOWLOADING ( 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 *subsystem* is `RECORDING_PROGRESS_REPORT`:

- When analyzer state is ANALYZERSTATE_IDLE, this parameter is not applicable.
- When analyzer state is `ANALYZERSTATE_WAITING_TRIGGER` or `ANALYZERSTATE_RECORDING_TRIGGERED`, this parameter shows analyzer memory utilization
- when analyzer state is `ANALYZERSTATE_UPLOADING_DATA,` this parameter shows the percent of data uploaded.
- When analyzer state is `ANALYZERSTATE_SAVING_DATA,` this parameter shows the percent of data saved.

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 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>
```

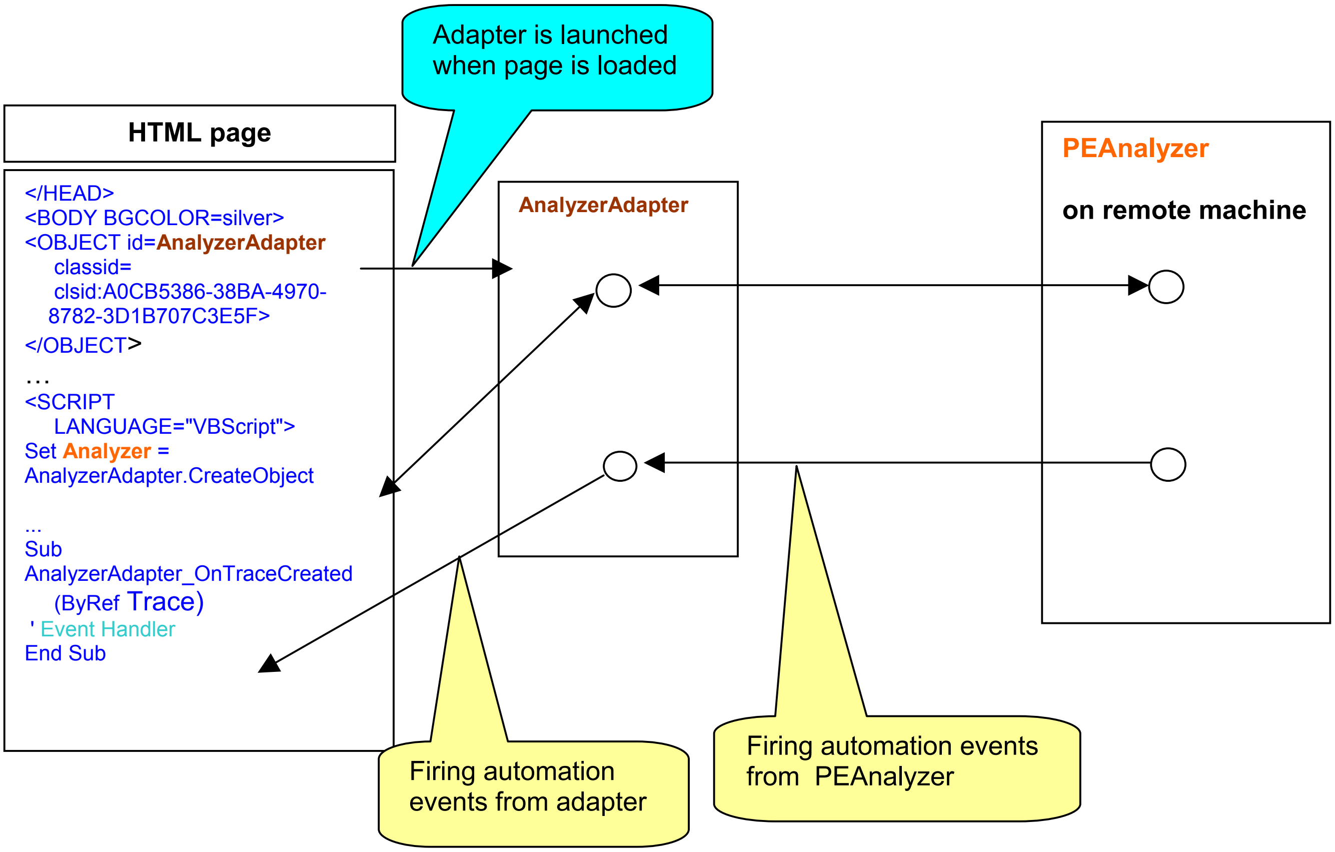
C++:

```
        #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 )

        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;
                default:
                        _tcscpy( status_buf, _T("Unknown") );
                        break;
                }
                _stprintf( m_RecordingStatus, _T("%s, done %ld%%"), status_buf, percent_done );
        }
```

# 12 CATCAnalyzerAdapter

*CATCAnalyzerAdapter* is an automation server that allows the launching and accessing of 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 PE*Tracer*™ analyzer server.

The following diagram illustrates how this functionality works:



Adapter is launched when page is loaded
HTML page
AnalyzerAdapter
PEAnalyzer
on remote machine
Firing automation events from adapter
Firing automation events from PEAnalyzer

```
</HEAD>
<BODY BGCOLOR=silver>
<OBJECT id=AnalyzerAdapter
    classid=
    clsid:A0CB5386-38BA-4970-
   8782-3D1B707C3E5F>
</OBJECT>
…
<SCRIPT
    LANGUAGE="VBScript">
Set Analyzer =
AnalyzerAdapter.CreateObject

...
Sub
AnalyzerAdapter_OnTraceCreated
    (ByRef Trace)
' Event Handler
End Sub
```

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: *IAnalazerAdapter*.

# 12.1IAnalyzerAdapter Interface

## 12.1.1 IAnalyzerAdapter::CreateObject

```
HRESULT CreateObject (
      [in] BSTR class_id,
      [in, optional] BSTR host_name,
      [out, retval] IDispatch** ppNewObj )
```

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 *classid* or *ProgId* (*clsid:297CD804-08F5-4A4F-B3BA-779B2654B27C* or *CATC.PETracer* for *PEAnalyzer* 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 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" )
        ...
```

## 12.1.2 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" )
        ...
```

## 12.1.3 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
                                   CreateOject()

        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
```

## 12.1.4 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 varable 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 PETracer !!!
               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>
```

# How to Contact LeCroy

| Type of Service | Contact |
|---|---|
| Call for technical support… | US and Canada: 1 (800) 909-2282<br>Worldwide: 1 (408) 727-6600 |
| Fax your questions… | Worldwide: 1 (408) 727-6622 |
| Write a letter … | LeCroy<br>Protocol Solutions Group<br>Customer Support<br>3385 Scott Blvd.<br>Santa Clara, CA 95054-3115 |
| Send e-mail… | support@catc.com |
| Visit LeCroy's web site… | http://www.lecroy.com/ |