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

Automation API

Version 2.3

Reference Manual

for

LeCroy USB*Tracer*™ and Advisor™

Version 2.3

Manual Version 2.3

July 2006

Document Disclaimer

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

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

Trademarks and Servicemarks

CATC Trace, USBTracer, USBTrainer, and *Advisor* are trademarks of LeCroy Corporation.

Microsoft and Windows are registered trademarks of Microsoft Inc.

All other trademarks are property of their respective companies.

Copyright

Copyright © 2006 LeCroy Corporation. All Rights Reserved.

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

Version

This API applies to version 2.3 of USB*Tracer*/*Trainer* and version 2.3 of Advisor.

Contents

1 Introduction

USB*Tracer*™ and Advisor™ Automation is an Application Program Interface (API) that allows users to create scripts or programs of commands and run these scripts or programs locally or remotely over a network. The name Automation is derived from the goal of allowing engineers to automate test procedures.

The USB*Tracer* and Advisor Automation API is composed of a command set that duplicates most of the functionality of the USB*Tracer* and Advisor Graphical User Interface. Automation is implemented through the use of scripts or programs that the user can write using late binding scripting languages such as VBScript and WSH or early binding languages such as C++. LeCroy provides examples of scripts written in WSH, VBScript, and C++.

Once an Automation script or program has been created, it can be run on the PC attached to or transmitted to the PC over a network from a remote location. Automation uses the Distributed Component Object Model (DCOM) protocol to transmit automation commands over a network. When run over a network, the Host Controller is configured as a DCOM server and the remote PC is configured as a DCOM client.

1.1 System Requirements

Automation is supported with USB*Tracer/Trainer* and Advisor Software Version 1.7 or higher. If you have an older version of the software, you must upgrade. You can get a new version of the USB*Tracer* and Advisor software from the LeCroy web site:

www.lecroy.com/support

You also need a copy of the USB*Tracer* and Advisor Automation software package. This is available from LeCroy.

1.2 Setting Up Automation for Local Use

If you intend to run Automation on the Analyzer Host Controller (the PC attached to the Analyzer), 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.

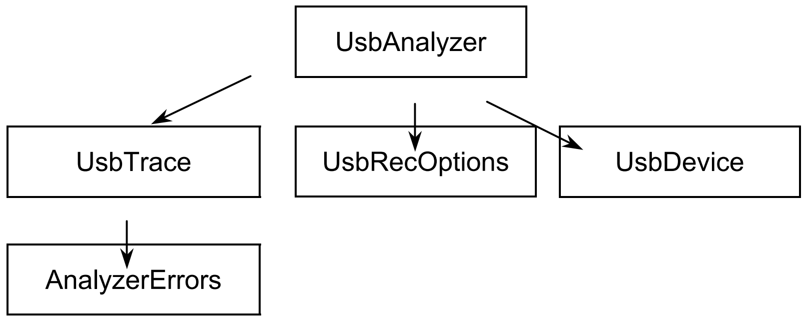
1.3 Setting Up Automation for Remote Use

If you intend to run automation remotely over a network, you must perform DCOM configuration. These steps are described in the Appendix.

The CATC UsbAnalyzer API exposes the following objects and interfaces:

Objects	Interfaces	Description
`UsbAnalyzer`	`IUsbAnalyzer`	Primary Analyzer interface
	`_IAnalyzerEvents`	Analyzer event source
`UsbTrace`	`IUsbTrace`	Trace file interface
`UsbRecOptions`	`IUsbRecOptions`	Recording options interface
`UsbDevice`	`IUsbDevice`	USB device interface
`AnalyzerErrors`	`IAnalyzerErrors`	Error collection interface

Only the `UsbAnalyzer` object is creatable at the top level (that is, via the `CoCreateInstance` call), instantiation of an object of other classes requires API calls. The following diagram represents the object dependencies:



All interfaces are dual interfaces, which allow simple use from typeless languages as well as from C++.

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

The examples of C++ code given in this document assume using the “import” technique of creating COM clients, using the corresponding **include** statement:

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

and creating appropriate wrapper classes in **.tli** and **.tlh** files by the compiler.

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

2 Primary Dual Interface for Analyzer

2.1 IUsbAnalyzer Dual Interface

`IUsbAnalyzer` interface is the primary interface for the `UsbAnalyzer` object. It derives from the `IAnalyzer` interface that implements the common functionality for all LeCroy Analyzers.

`Class ID:   0B179BB3-DC61-11d4-9B71-000102566088`
`App ID:     CATC.USBTracer`

2.1.1 IAnalyzer::GetVersion

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

Retrieves the current version of the specified subsystem.

Parameters

`version_type`	Subsystem whose version is requested; `EAnalyzerVersionType` enumerator has the following values: `ANALYZERVERSION_SOFTWARE` (0) Software `ANALYZERVERSION_BUSENGINE` (1) Bus engine `ANALYZERVERSION_FIRMWARE` (2) Firmware
`analyzer_version`	Current version of subsystem requested

Return values

ANALYZERCOMERROR_INVALIDVERSIONTYPE	Specified version type is invalid.
ANALYZERCOMERROR_ANALYZERNOTCONNECTED	Analyzer device is not connected.

Remarks

Example

```
WSH:
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       SwVersion = Analyzer.GetVersion(0)
       BEVersion = Analyzer.GetVersion(1)
       FwVersion = Analyzer.GetVersion(2)
       MsgBox "Software" & SwVersion & "BusEngine" & BEVersion & "Firmware" & FwVersion
```

C++:

```
        HRESULT      hr;
        IUsbAnalyzer*   poUsbAnalyzer;

        // create UsbAnalyzer object
        if ( FAILED( CoCreateInstance(
                 CLSID_UsbAnalyzer,
                 NULL, CLSCTX_SERVER,
                 IID_IUsbAnalyzer,
                 (LPVOID *)&poUsbAnalyzer ) )
                return;

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

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

2.1.2 IAnalyzer::GetSerialNumber

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

Retrieves the serial number of the Analyzer device.

Parameters

Return values

`ANALYZERCOMERROR_INVALIDVERSIONTYPE`	Specified version type is invalid.
`ANALYZERCOMERROR_ANALYZERNOTCONNECTED`	Analyzer device is not connected.

Remarks

Example

```
WSH:
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       MsgBox "Serial number: " & Analyzer.GetSerialNumber()


C++:

       HRESULT         hr;
       IUsbAnalyzer*   poUsbAnalyzer;

       // create UsbAnalyzer object
       if ( FAILED( CoCreateInstance(
                CLSID_UsbAnalyzer,
                NULL, CLSCTX_SERVER,
                IID_IUsbAnalyzer,
                (LPVOID *)&poUsbAnalyzer ) )
               return;

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

       TCHAR buffer[20];
       _stprintf( buffer, _T("Serial number: %X"),
                              HIBYTE(serial_number),
                              LOBYTE(serial_number) );
```

2.1.3 IAnalyzer::OpenFile

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

Opens a trace file.

Parameters

`file_name`	String providing the full pathname to the trace file
`trace`	Address of a pointer to the `UsbTrace` object primary interface

Return values

`ANALYZERCOMERROR_UNABLEOPENFILE`	Unable to open file.

Remarks

`UsbTrace` object is created via this method call, if call was successful.

Example

```
WSH:
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       Set Trace = Analyzer.OpenFile (CurrentDir & "Input\errors.usb")

C++:
       HRESULT         hr;
       IUsbAnalyzer*   poUsbAnalyzer;

       // create UsbAnalyzer object
       if ( FAILED( CoCreateInstance(
                CLSID_UsbAnalyzer,
                NULL, CLSCTX_SERVER,
                IID_IUsbAnalyzer,
                (LPVOID *)&poUsbAnalyzer ) )
               return;

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

       // query for VTBL interface
       IUsbTrace* usb_trace;
       hr = trace->QueryInterface( IID_IUsbTrace, (LPVOID *)&usb_trace );
       trace->Release();

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

2.1.4 IAnalyzer::StartGeneration

```
HRESULT StartGeneration (
      [in] BSTR gen_file_name,
      [in] long gen_mode,
      [in] long loop_count );
```

Starts traffic generation from the file.

Parameters

`gen_file_name`	String providing the full pathname to the generation file. If a valid name, the file is opened after any pre-existing Generation File is closed in accordance with the behavior set forth by Bit 31 in the **gen_mode** parameter. If NULL string, it just closes any existing Generation File.
`gen_mode`	Generation mode: Bit 0: 0 = bitstream; 1 = IntelliFrame Bit 31: 0 = Load **gen_file_name** only if: 1. Different than pre-existing Generation Filename OR 2. The **gen_file_name** file is already loaded but has been changed since the time it was loaded OR 3. The generation mode (bit 0) is different than the previous invocation. Leaving this bit set to 0 allows a file to be generated repeatedly without having to be re-parsed or downloaded to the Analyzer hardware, saving time. 1 = Reload **gen_file_name** unconditionally.
`loop_count`	Number of times to repeat: -1 = loop forever 1 through 16381 executes generation file that number of times.

Return values

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

Remarks

Used to load a **.utg** file and begin generating traffic.

Example

2.1.5 IAnalyzer::StopGeneration

```
HRESULT StopGeneration ( );
```

Stops any current generation in progress.

Return values

`ANALYZERCOMERROR_UNABLESTARTGENERATION`	Unable to stop generation (invalid state, etc.).

Remarks

Stop any current Traffic Generation.

Example

2.1.6 IAnalyzer::StartRecording

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

Starts recording with specified recording options.

Parameters

`ro_file_name`	String providing the full pathname to recording options file. If the parameter is omitted, then recording starts with the default recording options.

Return values

`ANALYZERCOMERROR_EVENTSINKNOTINSTANTIATED`	Event sink was not instantiated.
`ANALYZERCOMERROR_UNABLESTARTRECORDING`	Unable to start recording.

Remarks

After recording starts, this function returns. The Analyzer continues recording until it is finished or until a `StopRecording` method call is performed. During the recording, the events are sent to the event sink (see the `_IAnalyzerEvents` interface).

The recording options file is the file with extension **.rec** created by the UsbAnalyzer application. You can create such a file by selecting **Setup – Recording Options…** from the UsbAnalyzer application menu, changing the recording options in the dialog, and selecting the **Save** button.

Example

VBScript:

```
<OBJECT
        RUNAT=Server
        ID = Analyzer
        CLASSID = "clsid:0B179BB3-DC61-11d4-9B71-000102566088"
>
</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++:
```
        IUsbAnalyzer* usb_analyzer;
        BSTR          ro_file_name;

        . . .

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

2.1.7 IAnalyzer::StopRecording

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

Stops a recording started by the `StartRecording` method.

Parameters

`abort_upload`	TRUE, if caller wants to abort upload, no trace file is created. FALSE, if you want to upload the recorded trace

Return values

`ANALYZERCOMERROR_EVENTSINKNOTINSTANTIATED`	Event sink was not instantiated.
`ANALYZERCOMERROR_UNABLESTOPRECORDING`	Error stopping recording

Remarks

Stops recording started by the `StartRecording` method. The event is issued when recording is actually stopped (via `_IAnalizerEvents` interface), if the parameter of method call was FALSE.

Example

```
VBScript:

        <OBJECT
               RUNAT=Server
               ID = Analyzer
               CLASSID = "clsid:0B179BB3-DC61-11d4-9B71-000102566088"
        >
        </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++:
        IUsbAnalyzer* usb_analyzer;
        . . .
        try
        {
               usb_analyzer->StopRecording( FALSE )
        }
        catch ( _com_error& er)
        {
               if (er.Description().length() > 0)
               ::MessageBox( NULL, er.Description(),  _T("UsbAnalyzer client"), MB_OK );
               else
               ::MessageBox( NULL, er.ErrorMessage(), _T("UsbAnalyzer client"), MB_OK );
               return 1;
        }
```

2.1.8 IAnalyzer::MakeRecording

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

Makes a recording with the specified recording options file.

Parameters

`ro_file_name`	String providing the full pathname to recording options file; If the parameter is omitted, then recording starts with the default recording options.
`trace`	Address of a pointer to the `UsbTrace` object primary interface

Return values

Remarks

This method acts like `StartRecording` method but does not return until recording is completed.

`UsbTrace` object is created via this method call, if call was successful.

The recording options file is the file with extension **.rec** created by the UsbAnalyzer application. You can create such a file by selecting **Setup – Recording Options…** from the UsbAnalyzer application menu, changing the recording options in the dialog, and selecting the **Save** button.

Example

WSH:

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

C++:

```
IDispatch* trace;
IUsbAnalyzer* usb_analyzer;
BSTR       ro_file_name;
HRESULT    hr;

. . .

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

// query for VTBL interface
IUsbTrace* usb_trace;
hr = trace->QueryInterface( IID_IUsbTrace, (LPVOID *)&usb_trace );
trace->Release();
```

2.1.9 IAnalyzer::LoadDisplayOptions

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

Loads display options that apply for traces opened or recorded later.

Parameters

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

Return values

`ANALYZERCOMERROR_UNABLELOADDO`	Unable to load 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 files opened or recorded after this call.

Display options file is the file with extension **.opt** created by the UsbAnalyzer application. You can create such a file by selecting **Setup – Display Options…** from the UsbAnalyzer application menu, changing the display options in the dialog, and selecting the **Save** button.

Example

See `ITrace::ApplyDisplayOptions`.

2.1.10 IAnalyzer::GetRecordingOptions

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

Retrieves the primary interface for access to recording options.

Parameters

`recording_options`	Address of a pointer to the `UsbRecOptions` object primary interface

Return values

Remarks

`UsbRecOptions` object is created via this method call, if call was successful.

Example

```
WSH:
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       Set RecOptions = Analyzer.GetRecordingOptions

C++:

       HRESULT         hr;
       IUsbAnalyzer*   poUsbAnalyzer;

       // create UsbAnalyzer object
       if ( FAILED( CoCreateInstance(
                CLSID_UsbAnalyzer,
                NULL, CLSCTX_SERVER,
                IID_IUsbAnalyzer,
                (LPVOID *)&poUsbAnalyzer ) )
               return;

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

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

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

3 Primary Dual Interface for Trace

3.1 IUsbTrace Dual Interface

`IUsbTrace` interface is the primary interface for the `UsbTrace` object. It derives from the `ITrace` interface that implements the common functionality for all LeCroy Analyzers.

3.1.1 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.USBTracer")
        CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
        Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
        MsgBox "Trace name " & Trace.GetName

C++:
        IUsbTrace* usb_trace;

        . . .

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

3.1.2 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 display options file

Return values

`ANALYZERCOMERROR_UNABLELOADDO`	Unable to load 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 UsbAnalyzer application. You can create such a file by selecting **Setup – Display Options…** from the UsbAnalyzer application menu, changing the display options in the dialog, and selecting the **Save** button.

Example

```
WSH:
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       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.usb"

C++:

       IUsbTrace* usb_trace;
       TCHAR file_name[_MAX_PATH];

       . . .

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

3.1.3 ITrace::Save

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

Saves the trace into a file and allows you to save a range of packets.

Parameters

`file_name`	String providing the full pathname to file where trace is saved
`packet_from`	Beginning packet number when you are saving a range of packets. Value –1 means that the first packet of saved trace would be 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 saved trace would be the last packet of this trace.

Return values

`ANALYZERCOMERROR_UNABLESAVE`	Unable to save trace file.

Remarks

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

If packet range is specified and it is invalid (for example packet_to is more then last packet number in the trace, or packet_from is less then first packet number in the trace, or `packet_from` is more then `packet_to`) then packet range is adjusted automatically.

Example

```
WSH:
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       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.usb"

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

3.1.4 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 and allows you to export a range of packets.

Parameters

`file_name`	String providing the full pathname to file where trace is exported
`packet_from`	Beginning packet number when you are exporting a range of packets. Value –1 means that the first packet of exported trace would be 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 exported trace would be the last packet of this trace.

Return values

`ANALYZERCOMERROR_UNABLESAVE`	Unable to export trace file.

Remarks

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

If packet range is specified and it is invalid (for example packet_to is more then last packet number in the trace, or `packet_from` is less then first packet number in the trace, or `packet_from` is more then `packet_to`) then packet range is adjusted automatically.

Here is a snippet of export file for a Bluetooth Trace. A USB Trace file would look similar:

```
File E:\AnalyzerSw\Chief20\OfficialUTGFiles\SRP.usb.
From Packet #13 to Packet #24.

Packet#
_______|_______________________________________________________________________
13_____| Dir(-->) Suspend( 37.000 ms) Time-stamp(00016.0097 3086)
_______|_______________________________________________________________________
14_____| Dir(-->) Reset(  3.017 µs) Time-stamp(00016.0393 3090)
_______|_______________________________________________________________________
15_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(6) CRC5(0x09)
_______| EOP(266 ns) Time-stamp(00016.0393 3336)
_______|_______________________________________________________________________
16_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(7) CRC5(0x16)
_______| EOP(266 ns) Time-stamp(00016.0401 3336)
_______|_______________________________________________________________________
17_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(8) CRC5(0x06)
_______| EOP(266 ns) Time-stamp(00016.0409 3336)
_______|_______________________________________________________________________
18_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(9) CRC5(0x19)
_______| EOP(266 ns) Time-stamp(00016.0417 3336)
_______|_______________________________________________________________________
19_____| Dir(-->) F(S) Sync(00000001) OUT(0x87) ADDR(2) ENDP(3) CRC5(0x0C)
_______| EOP(266 ns) Time-stamp(00016.0417 3536)
_______|_______________________________________________________________________
20_____| Dir(-->) F(S) Sync(00000001) DATA0(0xC3)-BAD Data(8 bytes)
_______| CRC16(0xBB29) EOP(266 ns) Time-stamp(00016.0417 3726)
_______|_______________________________________________________________________
21_____| Dir(<--) F(S) Sync(00000001) ACK(0x4B) EOP(266 ns)
_______| Time-stamp(00016.0417 4256)
_______|_______________________________________________________________________
22_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(10) CRC5(0x1B)
_______| EOP(266 ns) Time-stamp(00016.0425 3336)
_______|_______________________________________________________________________
23_____| Dir(-->) F(S) Sync(00000001) SOF(0xA5) Frame #(11) CRC5(0x04)
_______| EOP(266 ns) Time-stamp(00016.0433 3336)
_______|_______________________________________________________________________
24_____| Dir(-->) Suspend(0 ns) Time-stamp(00016.0457 3466)
_______|_______________________________________________________________________
```

Example

WSH:

```
        Set Analyzer = WScript.CreateObject("CATC.USBTracer")
        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++:

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

3.1.5 ITrace::Close

```
    HRESULT Close ( );
```

Closes the trace.

Parameters

Return values

Remarks

Closes current trace, do not releases interface pointer. Call `IUnknown::Release` method right after this method call. No one `ITrace` method call succeeded after calling `ITrace::Close` method.

(Currently there is no need to call `ITrace::Close` directly since `IUnknown::Release` closes the trace.)

Example

3.1.6 ITrace::ReportFileInfo

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

Saves trace information into the specified text file.

Parameters

`file_name`	String with full pathname to file where trace information report is created

Return values

`ANALYZERCOMERROR_UNABLESAVE`	Unable to create trace information report.

Remarks

Creates trace information file if necessary. Stores trace information in specified file. Here is an example of data stored using this method call:

```
File name : data.usb
Comment :
Recorded on Channel number : 0
Number of packets : 22514
Trigger packet number : 0
Recorded with application version 1.70 ( Build 102 )
Analyzer Serial Number 00213
Traffic Generation enabled
   Firmware version 1.06 ( ROM 1.02 )
   BusEngine version 1.90
   BusEngine type 1
   UPAS Slot 1 Part# US005MA PlugIn ID: 0x01 Version 0x4
   UPAS Slot 2 Part# US005MG PlugIn ID: 0x04 Version 0x4

Number of markers : 1

Recording Options:
  Options Name: Default
  Recording Mode: Snapshot
  Buffer Size: 2.000 MB
  Post-trigger position: 50%
  Base filename & path: E:\AnalyzerSw\Chief20\Debug_UPA\data.usb
  Save External Signals No
  Auto-Merge No
  Truncate Data No
  Devices setup for On-the-Go operation
  2 DRD's: A device is named Camera, B device is named Printer
  Assumes first trace data is captured when A-Device is Host

  Channel 0 Trace Speed Recording Mode: Full Speed Only
  Channel 0 Recording Events:

  Channel 1 Trace Speed Recording Mode: Full Speed Only
  Channel 1 Recording Events:


Channel 0 is Full Speed.
Recorded on product: USBTracer
License information for the USBTracer unit, Serial Number 00213, used to record this
trace file :

Software maintenance hasn't been enabled.
```

Example

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

C++:
       IUsbTrace* usb_trace;
       TCHAR file_name[_MAX_PATH];

       . . .

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

3.1.7 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 file where error summary report is created

Return values

ANALYZERCOMERROR_UNABLESAVE	Unable to create trace information report.

Remarks

Creates error summary file if necessary. Stores error summary in specified file. Here is an example of data stored using this method call:

```
Error report for SRP.usb recording file.

_______|________________________________________________________________________
Bad PID (0):
_______|________________________________________________________________________
Bad CRC5 (0):
_______|________________________________________________________________________
Bad CRC16 (0):
_______|________________________________________________________________________
Bad Packet Length (0):
_______|________________________________________________________________________
Bad Stuff Bits (0):
_______|________________________________________________________________________
Bad EOP (0):
_______|________________________________________________________________________
Babble Start (0):
_______|________________________________________________________________________
Babble End (LOA) (0):
_______|________________________________________________________________________
Bad Frame Length (0):
_______|________________________________________________________________________
Bad Turnaround/Timeout (0):
_______|________________________________________________________________________
Bad Data Toggle (1):
_______| 0.20;
_______|________________________________________________________________________
Bad Frame/uFrame Number (0):
_______|________________________________________________________________________
Analyzer Internal Error (0):
_______|________________________________________________________________________
Last Byte Incomplete (0):
_______|________________________________________________________________________
```

Example

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

C++:
       IUsbTrace* usb_trace;
       TCHAR file_name[_MAX_PATH];

       . . .

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

3.1.8 ITrace::ReportTrafficSummary

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

Saves trace traffic summary information into the specified text file.

Parameters

`file_name`	String providing the full pathname to file where traffic summary report is created

Return values

`E_NOTIMPL`

Remarks

Not implemented yet.

Example

3.1.9 ITrace::GetPacket

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

Retrieves the raw packet representation.

Parameters

`packet_number`	Number of packet to retrieve
`packet`	Raw packet representation
`number_of_bits`	Number of bits in raw packet representation

Return values

`ANALYZERCOMERROR_INVALIDPACKETNUMBER`	Specified packet number is invalid

Remarks

The `packet` parameter has `VT_ARRAY | VT_VARIANT` actual automation type. Each element of this array has `VT_UI1` automation type. Since the last element of the array may contain extra data, you need to use the `number_of_bits` parameter to determine the actual packet data.

Example

```
VBScript:
        <OBJECT
                ID = Analyzer
                CLASSID = "clsid:0B179BB3-DC61-11d4-9B71-000102566088" >
        </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
                NumberOfBits = 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, NumberOfBits )
                        StatusText.innerText = "Packet ( " & NumberOfBits & " bits ): " &
                        PacketStr
                End If
        End Sub
        -->
        </SCRIPT>
```

```
C++:
        IUsbTrace* usb_trace;
        LONG packet_number;

        . . .

        VARIANT packet;
        VariantInit( &packet );
        long number_of_bits;
        try
        {
            number_of_bits = usb_trace->GetPacket( packet_number, &packet );
        }
        catch ( _com_error& er)
        {
            if (er.Description().length() > 0)
               ::MessageBox( NULL, er.Description(), _T("UsbAnalyzer client"), MB_OK );
            else
               ::MessageBox( NULL, er.ErrorMessage(),_T("UsbAnalyzer 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("UsbAnalyzer client"), MB_OK );
                    return 1;
                }
                if ( var.vt != ( VT_UI1) )
                {
    ::MessageBox( NULL, _T("Array of bytes expected"), _T("UsbAnalyzer client"), MB_OK );
                    return 1;
                }

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

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

3.1.10 ITrace::GetPacketsCount

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

Retrieves total number of packets in the trace.

Parameters

`number_of_packets`	Points to long value where number of packets in the trace is retrieved.

Return values

Remarks

Example

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

C++:
       IUsbTrace* usb_trace;

       . . .

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

3.1.11 ITrace::GetTriggerPacketNum

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

Retrieves trigger packet number.

Parameters

`packet_number`	Points to long value where trigger packet number is retrieved.

Return values

Remarks

Example

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

C++:
       See an example for ITrace::GetPacketsCount.
```

3.1.12 ITrace::AnalyzerErrors

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

Retrieves trace file errors.

Parameters

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

`0x00000001`	- Pid Error
`0x00000002`	- CRC5 Error
`0x00000004`	- CRC16 Error
`0x00000008`	- Packet Length Error
`0x00000010`	- Stuff Bit Error
`0x00000020`	– EOP Error
`0x00000040`	– Babble Start Error
`0x00000080`	– Babble End Error (LOA)
`0x00000100`	– Frame Length Error
`0x00000200`	– Handshake Timeout Error
`0x00000400`	– Analyzer Internal Error
`0x00000800`	– Data Toggle Error
`0x00001000`	– Microframe Error
`0x00002000`	– Short Byte Error (bits missing)

`analyzer_errors` Address of a pointer to the AnalyzerErrors object primary interface

Return values

`ANALYZERCOMERROR_INVALIDERROR` Invalid error type specified

Remarks

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

Example

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

C++:
        IUsbTrace* usb_trace;

        . . .

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

        . . .

        analyser_errors->Release();
```

4 Primary Dual Interface for Recording Options

4.1 IUsbRecOptions Dual Interface

`IUsbRecOptions` interface is the primary interface for the `IBRecOption` object. It derives from the `IRecOptions` interface that implements the common functionality for all LeCroy Analyzers.

4.1.1 IRecOptions::Load

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

Loads recording options from the specified file.

Parameters

`ro_file_name`	String providing 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.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.Load( CurrentDir & "Input\rec_options.rec" )
```

C++:

4.1.2 IRecOptions::Save

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

Saves recording options into the specified file.

Parameters

`ro_file_name`	String providing 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.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
' Do the changes of recording options here
RecOptions.Save( CurrentDir & "Input\rec_options.rec" )
```

C++:

4.1.3 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 the snapshot recording mode.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetRecMode 2 ' Event trigger
```

C++:

4.1.4 IRecOptions::SetBufferSize

```
HRESULT SetBufferSize (
      [in] long buffer_size );
```

Sets the size of buffer to record.

Parameters

`buffer_size`	Buffer size in bytes

Return values

`E_INVALIDARG`	Invalid buffer size was specified.

Remarks

The default setting is 1 MB.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetBufferSize 2*1024*1024 ' 2 MB
```

C++:

4.1.5 IRecOptions::SetPostTriggerPercentage

```
HRESULT SetPostTriggerPercentage (
      [in] short posttrigger_percentage );
```

Sets the post-trigger buffer size.

Parameters

`posttrigger_percentage`	Size of post trigger buffer in percents of the whole recording buffer (see 4.1.4)

Return values

`E_INVALIDARG`	Invalid percentage was specified.

Remarks

This method call has no effect if recording mode was set to `RMODE_SNAPSHOT` (see 4.1.3).
The default setting is 50%.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetPostTriggerPercentage 60 ' 60%
```

C++:

4.1.6 IRecOptions::SetTriggerBeep

```
HRESULT SetTriggerBeep (
      [in] BOOL beep );
```

Sets the flag to make a sound when a trigger occurs.

Parameters

`beep`	TRUE to beep when trigger occurs. FALSE to not beep when trigger occurs.

Return values

Remarks

The default state of the beeper is FALSE.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetTriggerBeep TRUE
```

C++:

4.1.7 IRecOptions::SetDataTruncate

```
HRESULT SetDataTruncate (
       [in] long length );
```

Sets the flag indicating that recorded data is to be truncated and sets the length of data to truncate.

Parameters

`length`	Length of data in bytes; cannot be less then 108

Return values

Remarks

By default, data is not truncated.

Example

WSH:

```
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       Set RecOptions = Analyzer.GetRecordingOptions
       RecOptions.SetDataTruncate 345 ' Truncate data that is more then 345 bytes long.
```

C++:

4.1.8 IRecOptions::SetAutoMerge

```
HRESULT SetAutoMerge (
      [in] BOOL auto_merge );
```

Sets the flag indicating that recorded traces on different channels are merged automatically after recording is done.

Parameters

`auto_merge`	TRUE performs merge automatically. FALSE does not perform merge.

Return values

Remarks

By default, automatic merge is not performed.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetAutoMerge TRUE
```

C++:

4.1.9 IRecOptions::SetSaveExternalSignals

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

Sets the flag indicating to save external signals.

Parameters

`save`	TRUE saves external signals. FALSE does not save external signals.

Return values

Remarks

By default, external signals are not saved.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetSaveExternalSignals TRUE
```

C++:

4.1.10 IRecOptions::SetTraceFileName

```
HRESULT SetTraceFileName (
      [in] BSTR file_name );
```

Sets the path to the file where the trace is stored after recording.

Parameters

`file_name`	String providing the full pathname to the file where recording is stored

Return values

Remarks

If 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.USBTracer")
       Set RecOptions = Analyzer.GetRecordingOptions
       ' Do the changes of recording options here.
       RecOptions.Save( CurrentDir & "Input\trace.usb" )

C++:
```

4.1.11 IRecOptions:: SetFilterPolarity

```
HRESULT SetFilterPolarity (
    [in] BOOL filter_out );
```

Sets whether to filter recording events in or out.

Parameters

`filter_out`	TRUE = filter out
	FALSE = filter in

Return values

`E_INVALIDARG`	Operation code and/or connection was specified.

Remarks

By default, all events are filtered out.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetFilterPolarity 0 ' filter in
```

C++:

4.1.12 IRecOptions::Reset

```
HRESULT Reset ( );
```

Resets recording options to its initial state.

Parameters

Return values

Remarks

For default values of recording options, see the remarks.

Example

WSH:

```
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
Set Analyzer = WScript.CreateObject("CATC.USBTracer")
Set RecOptions = Analyzer.GetRecordingOptions
RecOptions.SetRecMode 2             ' Event trigger
RecOptions.SetBufferSize 1024*1024  ' 1 MB
RecOptions.SetPostTriggerPercentage 60  ' 60%
. . .
RecOptions.Reset
```

C++:

5 Errors Collection Interface

5.1 IAnalyzerErrors dispinterface

This is a standard collection interface for collection of errors of a specified type (see ITrace::AnalyzerErrors). It has the following standard methods.

5.1.1 IAnalyzerErrors::get_Item

```
HRESULT get_Item(
      [in] long index,
      [out, retval] long* packet_number );
```

Parameters

`index`	Index of error in the collection
`packet_number`	Points to long value where error packet number is retrieved.

5.1.2 IAnalyzerErrors::get_Count

```
HRESULT get_Count(
      [out, retval] long* number_of_errors );
```

Parameters

`number_of_errors`	Points to long value where number of elements in the collection is retrieved.

Remarks

Example

```
WSH:
       ' Makes recording, and
       ' saves the portions of the recorded trace where "Bad VCRCs" errors occurred.
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Set Analyzer = WScript.CreateObject("CATC.USBTracer")
       Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
       Set Errors = Trace.AnalyzerErrors (16) ' Packet Length Error
       For Each ErrorPacketNumber In Errors
              ErrorFile = CurrentDir & "\Output\ PckLen_error_span_" &
       CStr(ErrorPacketNumber) & ".usb"
              Trace.Save ErrorFile, CInt(ErrorPacketNumber)-5, CInt(ErrorPacketNumber)+5
       Next
```

```
C++:
        IUsbTrace* usb_trace;

        . . .

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

        analyser_errors->Release();

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

6 Analyzer Events Callback Interface

6.1 _IAnalyzerEvents dispinterface

In order to retrieve the events from the UsbAnalyzer application, you must implement the `_IAnalyzerEvents` interface.

Since this interface is the default source interface for the `UsbAnalyzer` object, there is a very simple implementation from such languages as Visual Basic, VBA, VBScript, WSH, etc.

C++ implementation used in the examples below implements 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 which 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 establish the connection with the server, you need to advise your implementation of the event interface:

```
hr = CoCreateInstance( CLSID_UsbAnalyzer, NULL,
                  CLSCTX_SERVER, IID_IUsbAnalyzer, (LPVOID *)&m_poUsbAnalyzer );

m_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_poUsbAnalyzer, &m_poAnalyzerSink->m_libid,
       &m_poAnalyzerSink->m_iid, &m_poAnalyzerSink->m_wMajorVerNum,
       &m_poAnalyzerSink->m_wMinorVerNum);

 if ( FAILED(hr) )
       return 1;

// connect the sink and source, m_poUsbAnalyzer is the source COM object
hr = m_poAnalyzerSink->DispEventAdvise(m_poUsbAnalyzer, &m_poAnalyzerSink->m_iid);

if ( FAILED(hr) )
       return 1;
```

6.1.1 _IAnalyzerEvents::OnTraceCreated

```
HRESULT OnTraceCreated (
      [in] IDispatch* trace );
```

Fired when trace is created.
This event is a result of an IAnalyzer::StartRecording/ IAnalyzer::StopRecording method call.

Parameters

`trace`	Address of a pointer to the `UsbTrace` object primary interface

Return values

Remarks

Make sure the event handlers have `__stdcall` calling convention.

Example

VBScript:

```
        <OBJECT
                ID = Analyzer
                CLASSID = "clsid:0B179BB3-DC61-11d4-9B71-000102566088" >
        </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 )
        {
                IUsbTrace* usb_trace;
                HRESULT hr;
                hr = trace->QueryInterface( IID_IUsbTrace, (void**)&usb_trace );
                if (FAILED(hr))
                {
_com_error er(hr);
if (er.Description().length() > 0)
    ::MessageBox( NULL, er.Description(), _T("UsbAnalyzer client"), MB_OK );
else
    ::MessageBox( NULL, er.ErrorMessage(),_T("UsbAnalyzer client"), MB_OK );
return hr;
                }

                . . .

                return hr;
        }
```

6.1.2 _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

`state` Current Analyzer state; has the following values:
If `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

`percent_done` Shows the progress of currently performing operation, (valid only 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.

Return values

Remarks

Make sure the event handlers have `__stdcall` calling convention.

Example

```
VBScript:

        <OBJECT
                ID = Analyzer
                CLASSID = "clsid:0B179BB3-DC61-11d4-9B71-000102566088" >
        </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 );
```

7 CATCAnalyzerAdapter

Script languages, such as the Visual Basic (VB) and Javascript script languages used in HTML pages and Windows Script Host (WSH), can implement and automate exposed functionalities in client applications. However, such script engines cannot efficiently create automation objects dynamically, handle events, or control operation remotely. For example, an HTML page script language can only handle events from an automation object if the object is created with the <OBJECT> tag when the page is loaded. If the object is created at runtime, events from the object cannot be handled. An automation object can only be launched remotely if the name of the remote server is already known when the script client begins running. If the remote server is not yet known, automation objects cannot be launched.

The LeCroy USB Analyzer COM API includes an additional COM server, CATCAnalyzerAdapter, to provide full support for automation using script languages. The CATCAnalyzerAdapter automation server:

- Allows launching and accessing LeCroy analyzer automation servers dynamically at runtime.
- Allows launching and accessing LeCroy analyzer automation servers remotely, when the remote server has all necessary DCOM settings and permissions.
- Handles automation events from LeCroy analyzer automation servers.
- Overrides limitations imposed by the script engines used in HTML browsers and Windows Script Host.

The following diagram and the examples below show how the CATCAnalyzerAdapter automation server is used as an intermediate between an HTML page and the USBTracer analyzer server to create automation objects dynamically, handle events, and control operation remotely.

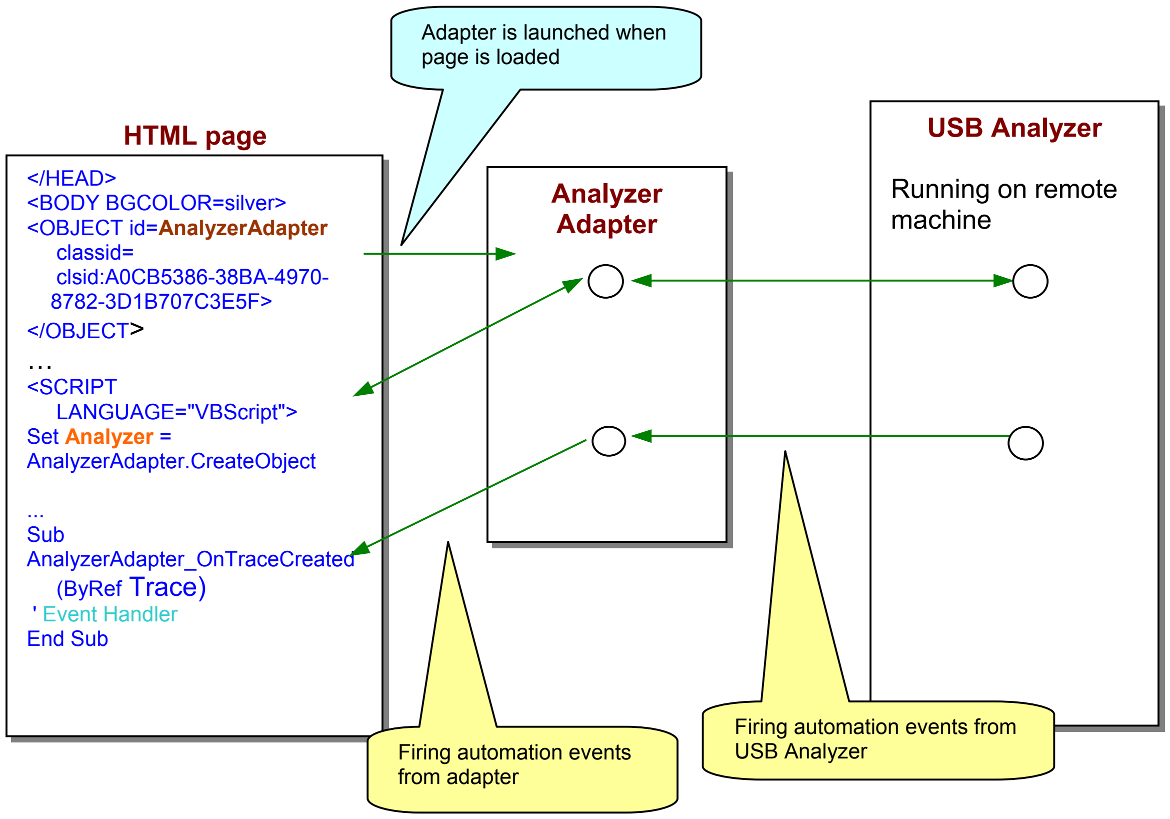

Adapter is launched when page is loaded
HTML page
</HEAD>
<BODY BGCOLOR=silver>
<OBJECT id=AnalyzerAdapter
classid=
clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
…
<SCRIPT
LANGUAGE="VBScript">
Set Analyzer =
AnalyzerAdapter.CreateObject
...
Sub
AnalyzerAdapter_OnTraceCreated
(ByRef Trace)
' Event Handler
End Sub
Analyzer Adapter
USB Analyzer
Running on remote machine
Firing automation events from adapter
Firing automation events from USB Analyzer

Figure 6-1 Analyzer Adapter

7.1 IAnalyzerAdapter Interface

```
Classid:                      A0CB5386-38BA-4970-8782-3D1B707C3E5F
ProgID:                       CATC.AnalyzerAdapter
COM server:                   CATCAnalyzerAdapter.exe

Primary default interface:    IAnalazerAdapter
```

7.1.1 IAnalyzerAdapter::CreateObject

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

This method instantiates a CATC Analyzer object on a local or remote machine and attaches it to the adapter.

Parameters

`class_id`	String representation of classid, or ProgId ("clsid: `0B179BB3-DC61-11d4-9B71-000102566088`" or "`CATC.USBTracer`" for CATC USB Analyzer)
`host_name`	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 CATC Analyzer COM servers can be instantiated through this method. The Detach method (see below) should be called when 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.USBTracer", RemoteServer.value)

                if Not Analyzer Is Nothing Then
                        window.status = "USBTracer connected"
                else
                        msg = "Unable to connect to USBTracer"
                        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.USBTracer", RemoteServer)

        Analyzer.StartRecording ( Analyzer.ApplicationFolder & "my.rec" )
        ...
```

7.1.2 IAnalyzerAdapter::Attach

```
HRESULT Attach([in] IDispatch* pObj);
```

This method attaches a CATC Analyzer object to the adapter.

Parameters

`pObj`	Pointer to the CATC Analyzer object to be attached.

Return values

Remarks

Only CATC Analyzer COM servers can be attached to the adapter. If some other Analyzer object was 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.USBTracer")
                ' VBScript function creates object locally.

                if Not Analyzer Is Nothing Then
                        AnalyzerAdapter.Attach Analyzer ' Attach analyzer to the adapter.

                        window.status = "USBTracer connected"
                else
                        msg = "Unable to connect to USBTracer"
                        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")
        ...
```

7.1.3 IAnalyzerAdapter::Detach

```
        HRESULT Detach();
```

This method detaches the CATC Analyzer object from the adapter.

Parameters

Return values

Remarks

This method detaches an Analyzer object from the adapter. This method does not guarantee that all resources associated with the detached object are 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.USBTracer", RemoteServer.value)

                if Not Analyzer Is Nothing Then
                        window.status = "USBTracer connected"
                else
                        msg = "Unable to connect to USBTracer"
                        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 = "USBTracer disconnected"
        End Sub
        -->
        </SCRIPT>
```

WSH:

```
' Create CATC analyzer adapter first.
Set AnalyzerAdapter = WScript.CreateObject("CATC.AnalyzerAdapter", "Analyzer_")

RemoteServer = "EVEREST"
Set Analyzer = AnalyzerAdapter.CreateObject("CATC.USBTracer", 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
```

7.1.4 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.

`pObj`	Pointer to the object validated
`pVal`	Pointer to the variable receiving result TRUE if the validated object can be attached FALSE otherwise.

Parameters

Return values

Remarks

Only CATC 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 USBTracer!!!
                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>
```

Appendix A. DCOM Configuration

USB*Tracer*™ and Advisor™ Automation can be run locally or remotely. This appendix describes how to configure Automation to run over a network. Automation uses Distributed Component Object Model (DCOM) to manage the transmission of automation commands over a network. When Automation is set up for remote usage, the Host Controller is configured as a DCOM server and the remote PC is configured as a DCOM client.

A.1. System Requirements

Automation is supported with USB*Tracer*/*Trainer* and Advisor Software Version 1.7 or higher. If you have an older version of the software, you must upgrade. You can get a new version of the software from the LeCroy web site:

www.LeCroy.com/support

A.2. Running Automation Locally

If you intend to run Automation on the Host Controller (the PC attached to the Analyzer), 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.

A.3. Setting Up Automation for Remote Use

If you intend to run automation remotely over a network, you must perform DCOM configuration. These steps are described below:

Summary of the Steps

1. Run the Microsoft™ utility **dcomcnfg** or **dcom98** on the Host Controller and configure the Host Controller so that it can be controlled by another device. For Windows™ 2000 and NT 4.0, use **dcomcnfg.** For Windows 98, use **dcom98. Dcom98** is not bundled with the operating system, so you must install it first.

2. Run **dcomcnfg** or **dcom98** on the remote PC.

3. Using **dcomcnfg** or **dcom98**, configure the remote PC so that it activates the Host Controller by default.

4. Test your DCOM configuration by running a script file on the remote PC. LeCroy provides some sample script files that you can use for this test.

5. If you write a program in C++, make sure that the Host Controller is registered on both the client and server machine.

A.4. DCOM Configuration for Host Controller

To configure DCOM on the Host Controller, you run a DCOM configuration utility called **dcomcnfg**. On Windows 2000 and Windows NT 4.0, DCOM and the **dcomcnfg** are bundled into the operating systems. On Windows 98, DCOM is not bundled into the operating system. You must first install DCOM before you can use **dcomcnfg** to configure the Analyzer for remote use.

When **dcomcnfg** is run, it presents a list of installed applications that support DCOM. To configure an application for DCOM, select an application from the list, then apply security and launch settings.

This appendix describes the process for configuring security and launch settings to the Analyzer for Windows 2000 and Windows NT 4.0 systems. If you are using Windows 98, you must first install DCOM before you can use **dcomcnfg**. The screens for Windows 98 are a little different from what you see in this appendix.

Summary of DCOM Server Configuration Steps

DCOM server configuration for the Host Controller involves three steps:

1. Configure DCOM Authentication to **Default**.

2. Configure DCOM launch permissions to **Everyone**.

3. Configure Run Identity with your login name.

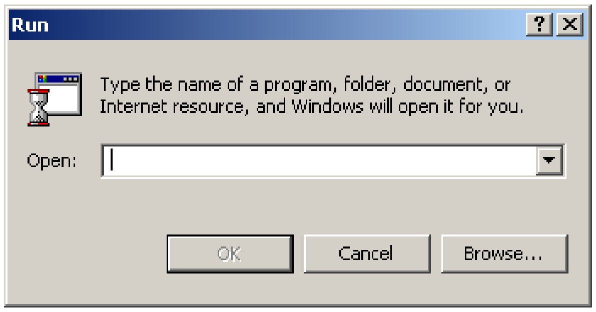
Configuring When and How Authentication Should Occur

The **dcomcnfg** utility presents a number of security options. In the steps that follow, configure the Analyzer application on the Host Controller to authenticate the user when the user first attempts a connection.

1. Click the Windows **Start** button.

2. Select **Run ...**

The Run dialog appears.

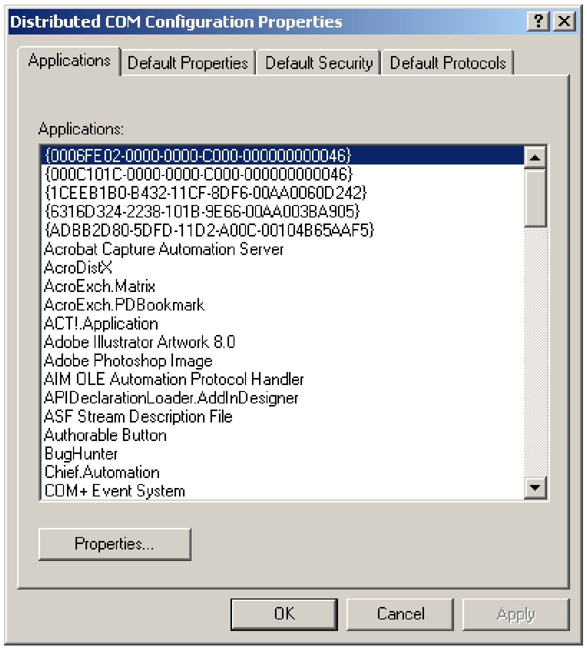


Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.

3. In the Open edit box within the Run dialog, type **dcomcnfg**.

4. Click **OK**.

The DCOM configuration dialog box opens.


Distributed COM Configuration Properties

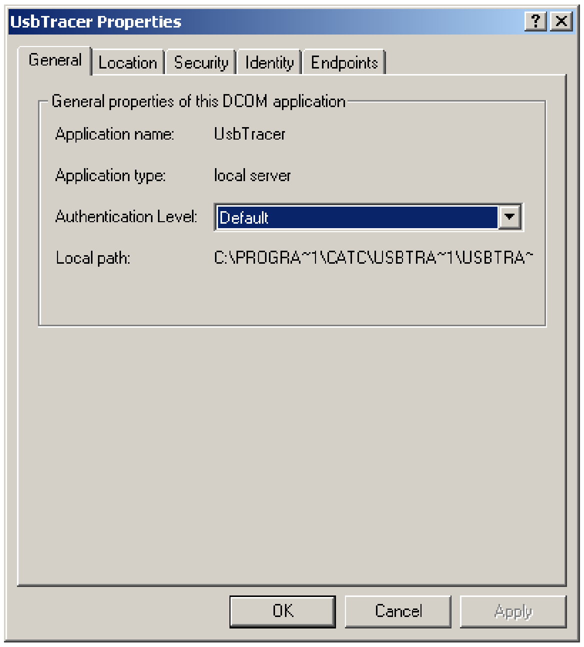
Applications:

{0006FE02-0000-0000-C000-000000000046}
{000C101C-0000-0000-C000-000000000046}
{1CEEB1B0-B432-11CF-8DF6-00AA0060D242}
{6316D324-2238-101B-9E66-00AA003BA905}
{ADBB2D80-5DFD-11D2-A00C-00104B65AAF5}
Acrobat Capture Automation Server
AcroDistX
AcroExch.Matrix
AcroExch.PDBookmark
ACT!.Application
Adobe Illustrator Artwork 8.0
Adobe Photoshop Image
AIM OLE Automation Protocol Handler
APIDeclarationLoader.AddInDesigner
ASF Stream Description File
Authorable Button
BugHunter
Chief.Automation
COM+ Event System

5. In the **Applications** tab, scroll down the list of applications and select **USBTracer/Trainer or Advisor**.

6. Click the **Properties** button.

The Properties dialog box opens. The options in this dialog box allow you to configure security on the selected application.


UsbTracer Properties

General properties of this DCOM application

Application name: UsbTracer

Application type: local server

Authentication Level: Default

Local path: C:\PROGRA~1\CATC\USBTRA~1\USBTRA~

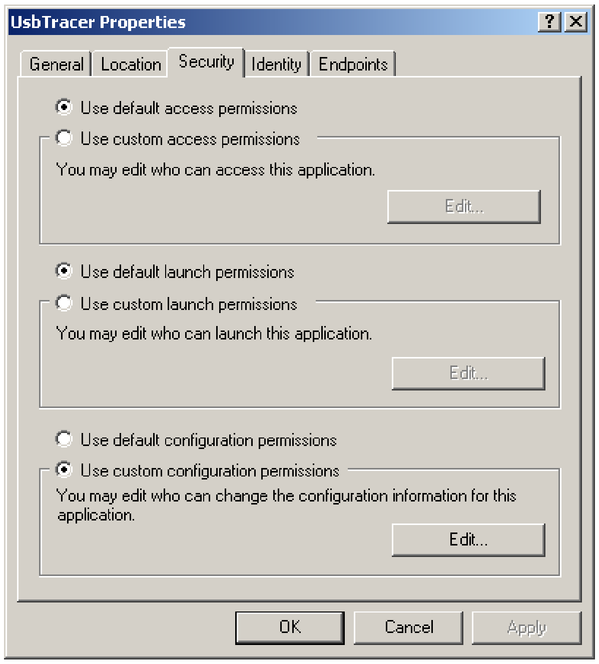
7. From the Authentication Level menu, select **Default**.

The Authentication Level menu lets you set packet-level security on communications for the selected application.

Configuring Access Permissions

Access permission determines who may execute commands on the application once it is running. In the following steps, you give everyone access to the application on the Host Controller.

1. With the Properties dialog box still displayed, select the **Security** tab.
The following settings display:


UsbTracer Properties

General | Location | Security | Identity | Endpoints

(●) Use default access permissions
( ) Use custom access permissions
You may edit who can access this application.
Edit...

(●) Use default launch permissions
( ) Use custom launch permissions
You may edit who can launch this application.
Edit...

( ) Use default configuration permissions
(●) Use custom configuration permissions
You may edit who can change the configuration information for this application.
Edit...

OK | Cancel | Apply

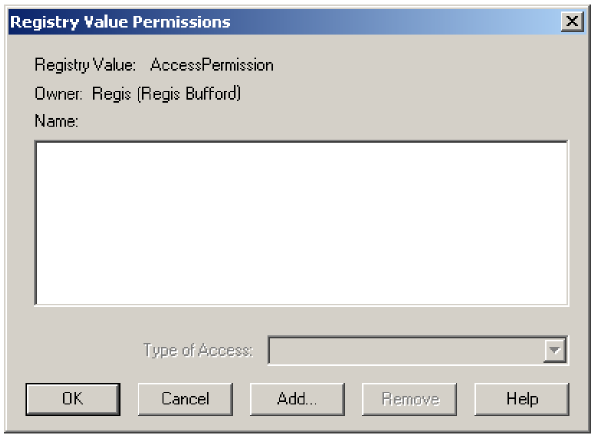
You see three options:

- **Access Permissions**: Determines who may execute commands on the application once it has been started
- **Launch Permissions**: Determines who may launch the application
- **Configuration Permissions**: Determines who may configure permissions for the application

2. Click the **User Custom Access** permissions option.
You are going to give all users permission to execute commands on the server.

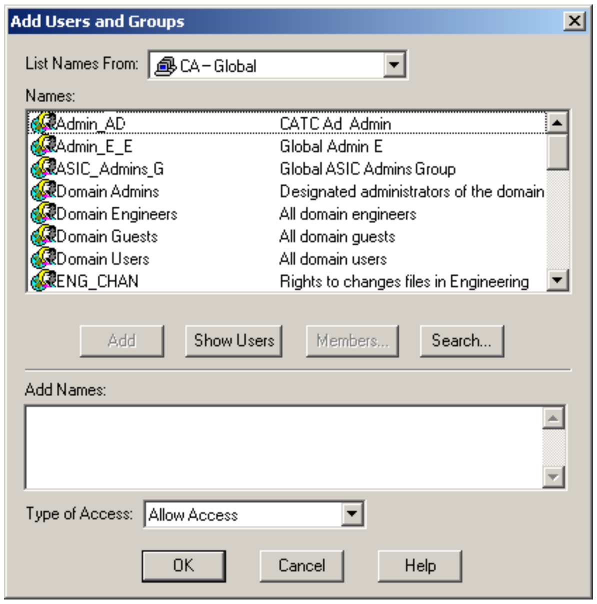
3. Click the **Edit ...** button.

The Registry Value Permissions dialog box appears.


Registry Value Permissions
Registry Value: AccessPermission
Owner: Regis (Regis Bufford)
Name:
Type of Access:
OK
Cancel
Add...
Remove
Help

4. Click the **Add ...** button.

The Add Users and Groups dialog box appears.



Admin_AD	CATC Ad Admin
Admin_E_E	Global Admin E
ASIC_Admins_G	Global ASIC Admins Group
Domain Admins	Designated administrators of the domain
Domain Engineers	All domain engineers
Domain Guests	All domain guests
Domain Users	All domain users
ENG_CHAN	Rights to changes files in Engineering

5. Select the group called **Everyone**.

6. Click the **Add** button.

7. Select the group **System**.

8. Click the **Add** button.

9. Click **OK**.

*The **Add Users** dialog box closes.*

10. Click **OK**.

The Add Users and Groups dialog box closes. The Registry Key Permissions dialog box remains on the screen.

11. Click **OK**.

The Registry Key Permissions dialog box closes. The Properties dialog box remains on the screen.

Configuring Launch Permissions

Launch Permissions control who can start an application. In the following steps, you give everyone permission to launch the application on the Host Controller.

1. In the Security tab of the server Properties dialog box, select the option **Custom Launch Permissions**.

2. Click the **Edit ...** button.
   The Registry Value Permissions dialog box appears.

3. Click the **Add ...** button.
   The Add Users and Groups dialog appears.

4. Select the group called **Everyone**.
   If you prefer, you can select individual user accounts instead of ***Everyone****.*

5. Click the **Add ...** button.

6. Click **OK**.
   The Add Users and Groups dialog box closes. The Registry Key Permissions dialog box remains on the screen.

7. Click **OK**.
   The Registry Key Permissions dialog box closes. The Properties dialog box remains on the screen.

Configuring Configuration Permissions

You are going to give everyone permission to configure permissions for the application on the Host Controller.

1. In the Security tab of the server Properties dialog box, select the option **User Custom Configuration Permissions**.

2. Click the **Edit ...** button.
   The Registry Value Permissions dialog box appears.

3. Click the **Add ...** button.
   The Add Users and Groups dialog appears.

4. Select the group called **Everyone**.
   If you prefer, you can select individual user accounts instead of ***Everyone****.*

5. Click the **Add ...** button.

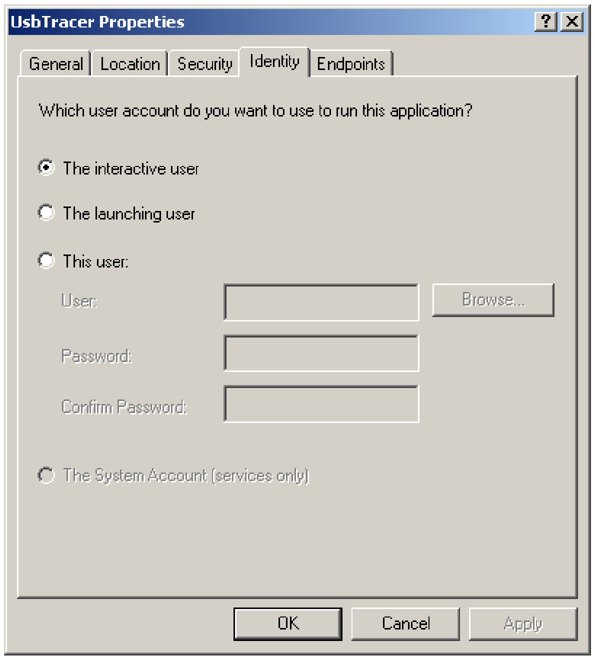
6. Click **OK**.
   The Add Users and Groups dialog box closes. The Registry Key Permissions dialog box remains on the screen.

7. Click **OK**.
   The Registry Key Permissions dialog box closes. The Properties dialog box remains on the screen.

Set User Run Permissions for Host Controller

If you want to create password-based security for individual users, perform the following steps on the Host Controller:

1. From the **Properties** dialog box, select the **Identity** tab.
The following screen displays:



Which user account do you want to use to run this application?

[✓ The interactive user]

The launching user

This user:

User:

Password:

Confirm Password:

The System Account (services only)

2. Make sure that the **Interactive User** option is selected.

A.5. DCOM Client Configuration

DCOM client configuration is a four-step process:

1. Set authentication to occur upon connection with the Host Controller.

2. Set the Host name or IP address of the server to which the client will connect.

3. Set commands to run on the Host Controller.

4. Set the network protocol over which DCOM will run.

Setting When and How Authentication Should Occur

You specify when and how the client should be authenticated by performing the following steps:

1. Click the Windows **Start** button.

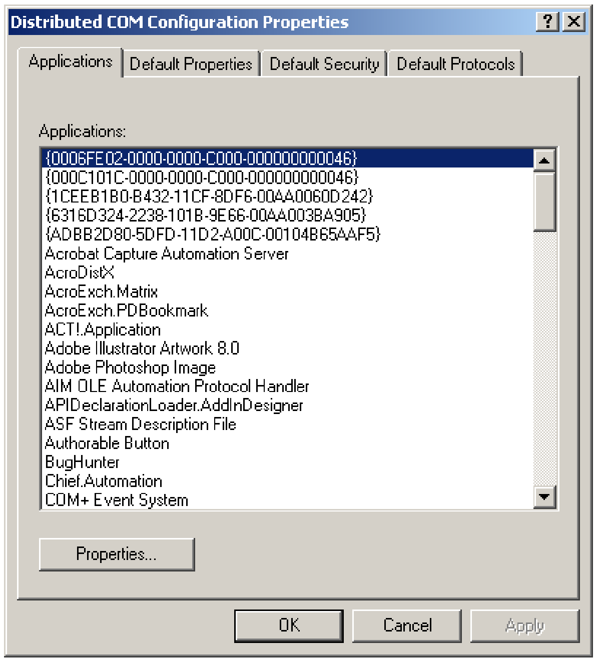
2. Select **Run ...**

   The Run dialog appears.

3. In the Open edit box within the Run dialog, type **dcomcnfg.**

4. Click **OK**.

   The following dialog box appears:



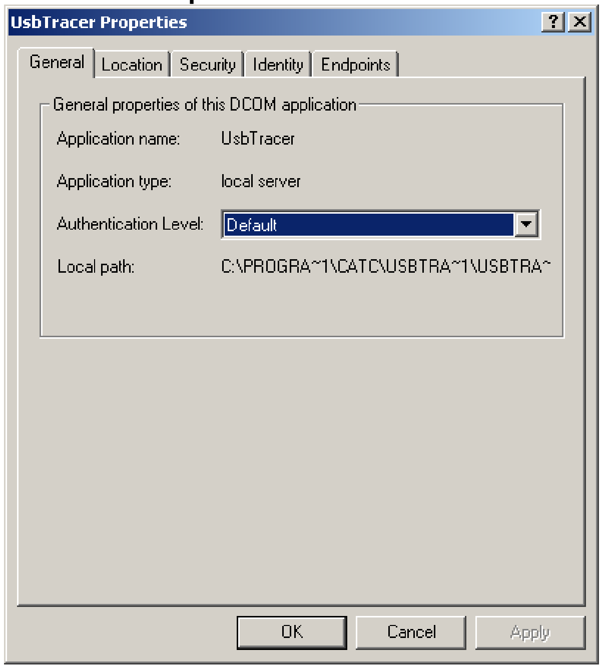
Applications:

{0006FE02-0000-0000-C000-000000000046}
{000C101C-0000-0000-C000-000000000046}
{1CEEB1B0-B432-11CF-8DF6-00AA0060D242}
{6316D324-2238-101B-9E66-00AA003BA905}
{ADBB2D80-5DFD-11D2-A00C-00104B65AAF5}
Acrobat Capture Automation Server
AcroDistX
AcroExch.Matrix
AcroExch.PDBookmark
ACT!.Application
Adobe Illustrator Artwork 8.0
Adobe Photoshop Image
AIM OLE Automation Protocol Handler
APIDeclarationLoader.AddInDesigner
ASF Stream Description File
Authorable Button
BugHunter
Chief.Automation
COM+ Event System

This dialog box presents a list of applications on the PC that support DCOM.

5. Select **USBTracer/Trainer or Advisor** from the list of applications.

6. Click the **Properties** button.



UsbTracer Properties

General | Location | Security | Identity | Endpoints

General properties of this DCOM application

Application name: UsbTracer

Application type: local server

Authentication Level: Default

Local path: C:\PROGRA~1\CATC\USBTRA~1\USBTRA~

7. Click the **Authentication** drop-down menu and select **Default**.

__Default__ tells the client to connect to the Host Controller using the Host Controller's Authentication Level.

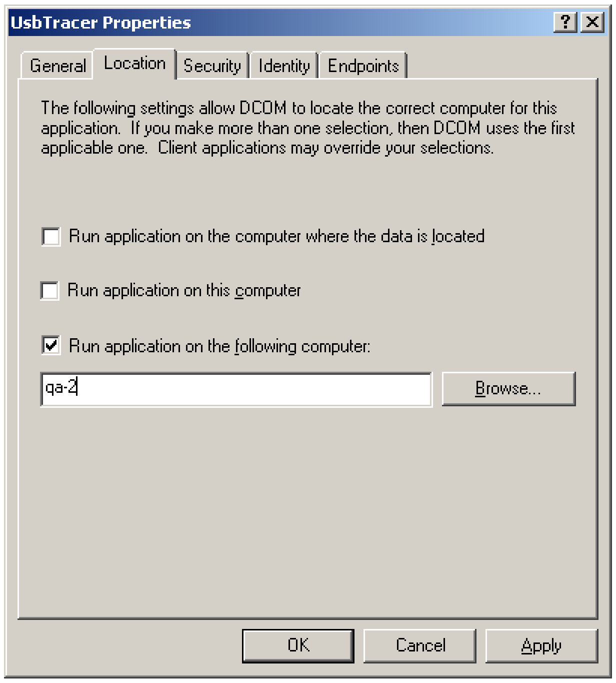
Set the Host Name or IP Address of the Host Controller

You must identify the device upon which the client will execute its commands.

Note: If you are writing a C/C++ program for automation, you can skip the following steps because you will specify the remote machines programmatically.

1. With the **Properties** dialog box still open, click the **Location** tab.

   The Location window displays.



The following settings allow DCOM to locate the correct computer for this application. If you make more than one selection, then DCOM uses the first applicable one. Client applications may override your selections.

☐ Run application on the computer where the data is located

☐ Run application on this computer

☑ Run application on the following computer:

qa-2

2. Select the option **Run application on the following computer**.

3. In the text box below the selected option, enter the **Host Name or IP address** of the Host Controller.

   *If you do not know the name of the Host Controller, you can browse to it using the **Browse ...** button.*

4. Click **OK**.

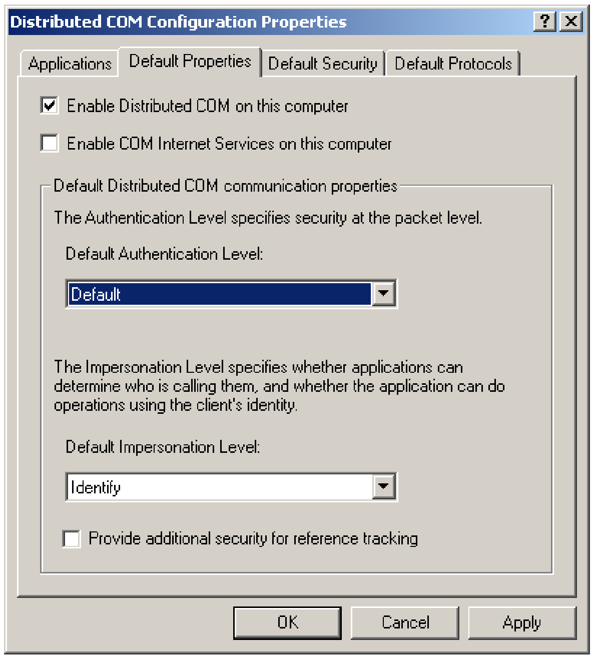
   The Properties dialog box closes. You see the Distributed COM Configuration Properties dialog box.

Set The Analyzer Commands to Execute on the Host Controller

The last configuration step is to set the "launch" location to **Server** for commands. This setting tells the client to execute commands remotely on the DCOM server rather than on itself.

1. On the Distributed Configuration Properties dialog box, click the tab marked **Default Properties**.

You should see the following settings. If the settings are not the same as those shown, change them to match the screenshot.



[✓] Enable Distributed COM on this computer

[ ] Enable COM Internet Services on this computer

Default Distributed COM communication properties

The Authentication Level specifies security at the packet level.

Default Authentication Level: Default

The Impersonation Level specifies whether applications can determine who is calling them, and whether the application can do operations using the client's identity.

Default Impersonation Level: Identify

[ ] Provide additional security for reference tracking

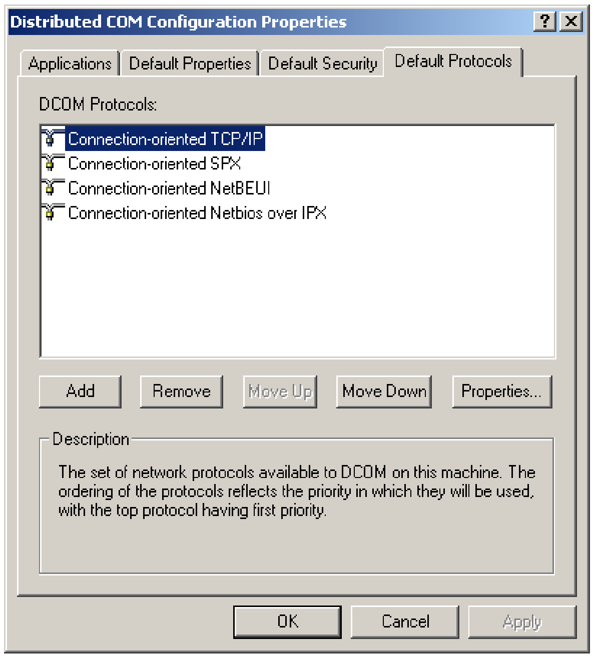
2. Select the option **Enable Distributed COM on this computer.**

You must perform this step on both computers - the DCOM server and DCOM client.

Set the Default Network Protocol

1. In the Distributed COM Configuration Properties dialog box, click the **Default Protocols** tab.

   *The following screen displays. TCP/IP should appear at the top of the list. If it does not, use the **Move Up** button to move it to the top of the list. If the protocol does not appear at all, you must add it using the **Add** button.*



DCOM Protocols:

Connection-oriented TCP/IP
Connection-oriented SPX
Connection-oriented NetBEUI
Connection-oriented Netbios over IPX

Description

The set of network protocols available to DCOM on this machine. The ordering of the protocols reflects the priority in which they will be used, with the top protocol having first priority.

2. Click **OK** to close the Distributed COM Configuration Properties dialog box.

   Your PC is now configured to run USBTracer/Trainer and Advisor Automation.

A.6. Samples for Automation

You can test your Automation setup by running a sample script file or executing one of the simple client applications included on the Automation installation package. The installation package contains three zipped directories:

- **cpp** - Contains C++ source code files. This directory includes the source code for an application called **analyzerclient.exe**. **Analyzerclient.exe** is a simple interface to the Automation API. **Analyzerclient.exe** lets you manage the Analyzer.
- **html** - Contains a directory with an HTML-based client application called **analyzer1.html** that is a simple interface to the Automation API.
- **wsh** - This is a directory containing Visual Basic script files. You can run these script files by double-clicking on them.

To test your Automation setup, execute **AnalyzerClient.exe** (after compiling it first) or **analzyer1.html**, or double-click one of the **wsh** files. If your setup is correct, execute a LeCroy sample script on the client PC that connects to the Host Controller. Sample script files can be found on the Automation Installation diskette.

Testing Your Automation Setup with AnalyzerClient.exe

The following steps show you how to test your automation setup using **AnalyzerClient.exe**. As mentioned above, you must compile this first from the source code in the **cpp** directory. Please refer to the ReadMe file in the Automation package for information about compiling the application.

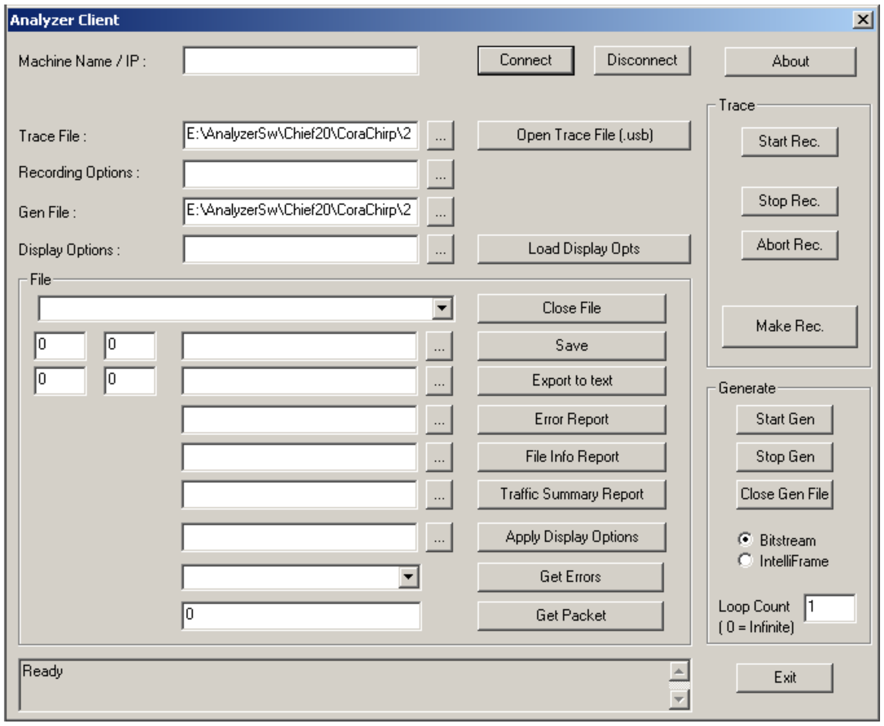
In the steps below, you execute **analyzerclient.exe** on a remote PC and establish a connection with the Host Controller PC. Then, you remotely load a trace file on the Host Controller.

If you are running Windows 98 on the Host Controller, run the Interface software on the Host Controller before following the steps below. Windows 98 has security issues that can be bypassed by first opening the software.

1. Browse to the directory in which you compiled **AnalyzerClient.exe**.

2. Double-click **AnalyzerClient.exe**.

The application starts. If you prefer, you can copy the executable to your hard drive and launch it from there.



Trace File : E:\AnalyzerSw\Chief20\CoraChirp\2
Gen File : E:\AnalyzerSw\Chief20\CoraChirp\2
(● Bitstream ○ IntelliFrame)
Loop Count (0 = Infinite) 1
Ready

3. Enter the **IP address or Host Name** of the Host Controller.

4. Press **Connect**.

The status message at the bottom of the window should change from **Ready** to a message that reads something like **USBTracer: Version 1.70**. This change indicates that a connection has been established to the Host Controller.

When you have completed this step, the Interface software automatically launches on the Host Controller.

If the message "The RPC server is unavailable" appears, check the network connections between the two PCs. Also try using the IP address in place of the Host Name (assuming that you had originally used the Host Name).

If the message "Access denied" appears, make sure that the account that you are using on the Host Controller is the same as the one you are using on the client PC and that both have Administrator privileges.

5. Click the ... button next to the **Trace File** text box. You are going to browse for a trace file. If you are going to run the commands over a network, you must supply the network path.

6. Browse for a trace file of your choice. If you do not have a trace file on your PC, you can select one from an Installation diskette. When you have selected a file, its name should appear in the **Trace File** text box.

7. Press the button marked **Open File**. A message should appear at the bottom of the window listing the trace name, file size (in packets) and trigger position.

Appendix B. How to Contact **LeCroy**

Type of Service	**Contact**	
Call for technical support…	US and Canada:	1 (800) 909-2282
	Worldwide:	1 (408) 727-6600
Fax your questions…	Worldwide:	1 (408) 727-6622
Write a letter …	LeCroy Corporation Customer Support 2403 Walsh Avenue Santa Clara, CA 95051-1302	
Send e-mail…	support@CATC.com	
Visit LeCroy's web site…	http://www.LeCroy.com/	