



**Computer Access Technology  
Corporation**

2403 Walsh Avenue, Santa Clara, CA 95051-1302  
Tel: +1/408.727.6600 Fax: +1/408.727.6622

# USB Tracer and Advisor Application Programming Interface User' s Manual

USBTracer and Advisor API Manual Version 1.0

**For USBTracer and Advisor Software Version 1.7 or Higher**

8 February, 2002

## 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.

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

## Trademarks and Servicemarks

*CATC, USBTracer and Advisor Automation* are trademarks of Computer Access Technology Corporation.

*Microsoft, Windows, Windows NT, Windows 98, Windows 2000, Windows ME, and Windows XP* are registered trademarks of Microsoft Inc.

All other trademarks are property of their respective companies.

## Copyright

Copyright © 2002, Computer Access Technology Corporation (CATC); 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 1.7 of *USBTracer/Trainer* and version 1.7 of *Advisor*.

# Contents

<b>1</b>	<b>Introduction.....</b>	<b>5</b>
1.1	System Requirements .....	5
1.2	Setting Up Automation for Local Use.....	5
1.3	Setting Up Automation for Remote Use .....	5
<b>2</b>	<b>Primary dual interface for analyzer.....</b>	<b>7</b>
2.1	IUsbAnalyzer dual interface .....	7
2.2	IAnalyzer::GetVersion.....	8
2.3	IAnalyzer::GetSerialNumber.....	10
2.4	IAnalyzer::OpenFile .....	11
2.5	IAnalyzer::StartGeneration.....	13
2.6	IAnalyzer::StopGeneration.....	14
2.7	IAnalyzer::StartRecording.....	15
2.8	IAnalyzer::StopRecording.....	17
2.9	IAnalyzer::MakeRecording .....	19
2.10	IAnalyzer::LoadDisplayOptions.....	20
2.11	IAnalyzer::GetRecordingOptions.....	21
<b>3</b>	<b>Primary dual interface for trace.....</b>	<b>22</b>
3.1	IUsbTrace dual interface .....	22
3.2	ITrace::GetName .....	23
3.3	ITrace::ApplyDisplayOptions .....	24
3.4	ITrace::Save.....	25
3.5	ITrace::ExportToText.....	26
3.6	ITrace::Close.....	28
3.7	ITrace::ReportFileInfo.....	29
3.8	ITrace::ReportErrorSummary.....	31
3.9	ITrace::ReportTrafficSummary .....	33
3.10	ITrace::GetPacket .....	34
3.11	ITrace::GetPacketsCount.....	37
3.12	ITrace::GetTriggerPacketNum .....	38
3.13	ITrace::AnalyzerErrors .....	39
<b>4</b>	<b>Primary dual interface for recording options.....</b>	<b>41</b>
4.1	IUsbRecOptions dual interface.....	41
4.2	IRecOptions::Load.....	42
4.3	IRecOptions::Save .....	43
4.4	IRecOptions::SetRecMode .....	44

4.5	IRecOptions::SetBufferSize.....	45
4.6	IRecOptions::SetPostTriggerPercentage .....	46
4.7	IRecOptions::SetTriggerBeep.....	47
4.8	IRecOptions::SetDataTruncate .....	48
4.9	IRecOptions::SetAutoMerge .....	49
4.10	IRecOptions::SetSaveExternalSignals.....	50
4.11	IRecOptions::SetTraceFileName .....	51
4.12	IRecOptions:: SetFilterPolarity.....	52
4.13	IRecOptions::Reset .....	53
<b>5</b>	<b>Errors collection interface.....</b>	<b>54</b>
5.1	IAnalyzerErrors dispinterface.....	54
5.2	IAnalyzerErrors::get_Item .....	55
5.3	IAnalyzerErrors::get_Count.....	56
<b>6</b>	<b>Analyzer events callback interface .....</b>	<b>58</b>
6.1	_IAnalyzerEvents dispinterface.....	58
6.2	_IAnalyzerEvents::OnTraceCreated.....	59
6.3	_IAnalyzerEvents::OnStatusReport .....	60

# 1 Introduction

USBTracer 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 USBTracer and Advisor Automation API is composed of a command set that duplicates most of the functionality of the USBTracer 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++. CATC 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 the 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 remote PC is configured as a DCOM client.

## 1.1 System Requirements

Automation is supported with **USBTracer/Trainer and Advisor Software Version 1.7** or higher. If you have an older version of the software, you will need to upgrade. You can get new version of USBTracer and Advisor software from the CATC web site:

[www.catc.com/support](http://www.catc.com/support)

You will also need a copy of the USBTracer and Advisor Automation software package. This is available from CATC.

## 1.2 Setting Up Automation for Local Use

If you intend to run Automation on the Analyzer Host Controller (i.e., 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 will run the analyzer.

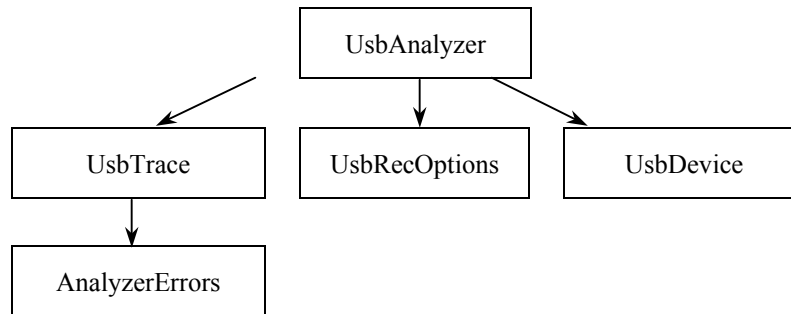
## 1.3 Setting Up Automation for Remote Use

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

CATC UsbAnalyzer API exposes the following objects and interfaces:

Objects	Interfaces	Description
UsbAnalyzer	IUsbAnalyzer _IAnalyzerEvents	primary analyzer interface 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 `CoCreateInstance` call), instantiation of an object of other classes requires API calls. The following diagram represents objects dependence:



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

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

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

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

and appropriate wrapper classes are created in `.tli`, `.tlh` files by compiler

The sample of WSH, VBScript, C++ client applications are provided.

## 2 Primary dual interface for analyzer

### 2.1 IUsbAnalyzer dual interface

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

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

## 2.2 IAnalyzer::GetVersion

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

Retrieves the current version of specified subsystem

### Parameters

version\_type - subsystem which 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) );

```

## 2.3 IAnalyzer::GetSerialNumber

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

Retrieves serial number of 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) );
```

## 2.4 IAnalyzer::OpenFile

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

Opens trace file

### Parameters

file_name	- string providing the full pathname to trace file
trace	- address of a pointer to the <code>UsbTrace</code> 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;
```

## 2.5 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 generation file. If a valid name, the file will be 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 will just close 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 much time.  
    1 – reload gen\_file\_name unconditionally
- loop\_count - number of times to repeat: -1 – loop forever, 1 through 16381 executes the 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

## 2.6 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

## 2.7 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 default recording options

### Return values

<code>ANALYZERCOMERROR_EVENTSINKNOTINSTANTIATED</code>	- event sink was not instantiated
<code>ANALYZERCOMERROR_UNABLESTARTRECORDING</code>	- unable to start recording

### Remarks

After recording starts this function will return. Analyzer continues recording until it is finished or until “StopRecording” method call is performed. During the recording the events are send to event sink (see `_IAnalyzerEvents` interface).

Recording options file is the file with extension .rec created by UsbAnalyzer application. You can create such file when you select “Setup – Recording Options...” from UsbAnalyzer application menu, change the recording options in the appeared dialog and select “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;
}
```



## 2.8 IAnalyzer::StopRecording

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

Stops recording started by `StartRecording` method

### Parameters

`abort_upload` – TRUE, if caller wants to abort upload, no trace file will be created, FALSE if want to upload recorded trace

### Return values

<code>ANALYZERCOMERROR_EVENTSINKNOTINSTANTIATED</code>	- event sink was not instantiated
<code>ANALYZERCOMERROR_UNABLESTOPRECORDING</code>	- error stopping recording

### Remarks

Stops recording started by '`StartRecording`' method. The event will be issued when recording is actually stopped (via `_IAnalyzerEvents` 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;  
}
```

## 2.9 IAnalyzer::MakeRecording

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

Makes recording with 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 default recording options

`trace` - address of a pointer to the `UsbTrace` object primary interface

### Return values

### Remarks

This method acts like 'StartRecording' method but will not return until recording is completed. `UsbTrace` object is created via this method call, if call was successful.

Recording options file is the file with extension .rec created by UsbAnalyzer application. You can create such file when you select "Setup – Recording Options..." from UsbAnalyzer application menu, change the recording options in the appeared dialog and select "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();
```

## 2.10 IAnalyzer::LoadDisplayOptions

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

Loads display options that will apply for trace 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 will apply only on trace file opened or recorded after this call.

Display options file is the file with extension .opt created by UsbAnalyzer application. You can create such file when you select “Setup – Display Options...” from UsbAnalyzer application menu, change the display options in the appeared dialog and select “Save...” button.

### Example

See ITrace::ApplyDisplayOptions

## 2.11 IAnalyzer::GetRecordingOptions

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

Retrieves 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 `UsbTrace` object. It derives from `ITrace` interface that implements the common functionality for all CATC analyzers.

## 3.2 ITrace::GetName

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

Retrieves trace name

### Parameters

trace\_name - the 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 );
```

## 3.3 ITrace::ApplyDisplayOptions

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

Applies 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.

Display options file is the file with extension .opt created by UsbAnalyzer application. You can create such file when you select “Setup – Display Options...” from UsbAnalyzer application menu, change the display options in the appeared dialog and select “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;
}
```



## 3.4 ITrace::Save

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

Saves trace into file, allows 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 will be 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;
}
```

## 3.5 ITrace::ExportToText

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

Exports trace into text file, allows 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 will be 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;
}
```

## 3.6 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 will succeeded after calling `ITrace::Close` method.

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

### Example

## 3.7 ITrace::ReportFileInfo

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

Saves trace information into specified text file

### Parameters

file\_name - string providing the 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;
}
```

## 3.8 ITrace::ReportErrorSummary

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

Saves trace error summary information into 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;
    }

```



## 3.9 ITrace::ReportTrafficSummary

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

Saves trace traffic summary information into 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

## 3.10 ITrace::GetPacket

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

Retrieves 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

“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 number\_of\_bits parameter to determine 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
    End Sub
```

```
        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
);
}

```

## 3.11 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 );
```

## 3.12 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

## 3.13 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 `IBRecOption` object. It derives from `IRecOptions` interface that implements the common functionality for all CATC analyzers.

## 4.2 IRecOptions::Load

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

Loads recording options from 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++:

## 4.3 IRecOptions::Save

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

Saves recording options into 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 specified file does not exists it will be create, if it exists it will be 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++:

## 4.4 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:

<code>RMODE_SNAPSHOT</code>	<code>( 0 )</code>	- snapshot recording mode
<code>RMODE_MANUAL</code>	<code>( 1 )</code>	- manual trigger
<code>RMODE_USE_TRG</code>	<code>( 2 )</code>	- event trigger

### Return values

`E_INVALIDARG` - invalid recording mode was specified

### Remarks

The default setting of recording options is a snapshot recording mode.

### Example

WSH:

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

C++:

## 4.5 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 1Mb.

### Example

WSH:

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

C++:

## 4.6 IRecOptions::SetPostTriggerPercentage

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

Sets the posttrigger buffer size

### Parameters

posttrigger\_percentage - the size of post trigger buffer in percents of the whole recording buffer ( see 4.5 )

### Return values

E\_INVALIDARG - invalid percentage was specified

### Remarks

This method call has no effect if recording mode was set to RMODE\_SNAPSHOT ( see 4.4 ).  
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++:

## 4.7 IRecOptions::SetTriggerBeep

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

Sets the flag indicating to make a sound when trigger occurs

### Parameters

beep - TRUE – beep when trigger occurs, FALSE – do 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++:

## 4.8 IRecOptions::SetDataTruncate

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

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

### Parameters

`length` - length of data in bytes, could not be less than 108

### Return values

### Remarks

By default data is not truncating.

### 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 than 345 bytes
```

long

C++:



## 4.9 IRecOptions::SetAutoMerge

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

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

### Parameters

auto\_merge      - TRUE – perform merge automatically; FALSE – do not perform merge

### Return values

### Remarks

By default automatic merge is not performing

### Example

WSH:

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

C++:

## 4.10 IRecOptions::SetSaveExternalSignals

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

Sets the flag indicating to save external signals

### Parameters

`save` - TRUE – save external signals, FALSE – do not save external signals

### Return values

### Remarks

By default external signals are not saved.

### Example

WSH:

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

C++:

## 4.11 IRecOptions::SetTraceFileName

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

Sets the path to the file where trace will be stored after recording

### Parameters

file\_name - string providing the full pathname to the file where recording will be stored

### Return values

### Remarks

If specified file does not exist it will be created, if it exists it will be 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++:

## 4.12 IRecOptions:: SetFilterPolarity

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

Sets the whether to filter in or out the recording events

### 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++:

## 4.13 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 ' 1Mb
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 specified type (see `ITrace::AnalyzerErrors`). It has the following standard methods:

## 5.2 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

## 5.3 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, 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 UsbAnalyzer application you must implement `_IAnalyzerEvents` interface.

Since this interface is default source interface for `UsbAnalyzer` object there is very simple implementation

from such languages like Visual Basic, VBA, VBScript, WSH etc.

C++ implementation used in the examples below utilizes implements a sink object by deriving from `IDispatchImpl` 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 IDispatchImpl<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 established the connection with 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 IDispatchImpl
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->DispatchEventAdvise(m_poUsbAnalyzer, &m_poAnalyzerSink->m_iid);

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

## 6.2 IAnalyzerEvents::OnTraceCreated

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

Fired when trace is created; this event is a result of 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;
}
```

## 6.3 **\_IAnalyzerEvents::OnStatusReport**

```
HRESULT OnStatusReport (
    [in] short subsystem,
    [in] short state,
    [in] long percent_done );
```

Fired when there is a change in analyzer's state or there is a change in progress (percent\_done) of 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 );
}

```

# Appendix A:

## DCOM Configuration

USBTracer 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 remote PC is configured as a DCOM client.

### A.1. System Requirements

Automation is supported with **USBTracer/Trainer and Advisor Software Version 1.7** or higher. If you have an older version of the software, you will need to upgrade. You can get new version of software from the CATC web site:

[www.cafc.com/support](http://www.cafc.com/support)

### A.2. Running Automation Locally

If you intend to run Automation on the Host Controller (i.e., 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 will run the analyzer.

### A.3. Setting Up Automation for Remote Use

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

*A 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 - you will need to install it first.
2. Run *dcomcnfg* or *dcom98* on the remote PC.
3. Using *dcomcnfg* or *dcom98*, configure the remote PC so that it will activate the Host Controller by default.
4. Test your DCOM configuration by running a script file on the remote PC. CATC 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 will 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 will need to first install DCOM before you can use *dcomcnfg* to configure The Analyzer for remote use.

When *dcomcnfg* is run, it will present 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 will need to first install DCOM before you can use *dcomcnfg*. The screens for Windows 98 will be a little different from what you see in this appendix.

### *Summary of DCOM Server Configuration Steps*

DCOM server configuration for the Host Controller involves 3 steps:

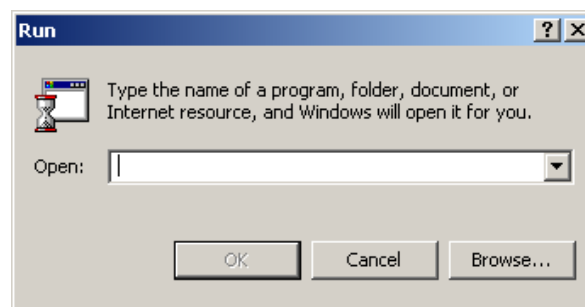
1. Configure DCOM Authentication to "Default."
2. Configure DCOM launch permissions to "Everyone."
3. Configure Run Identity with your log in name.

### **Configuring When & How Authentication Should Occur**

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

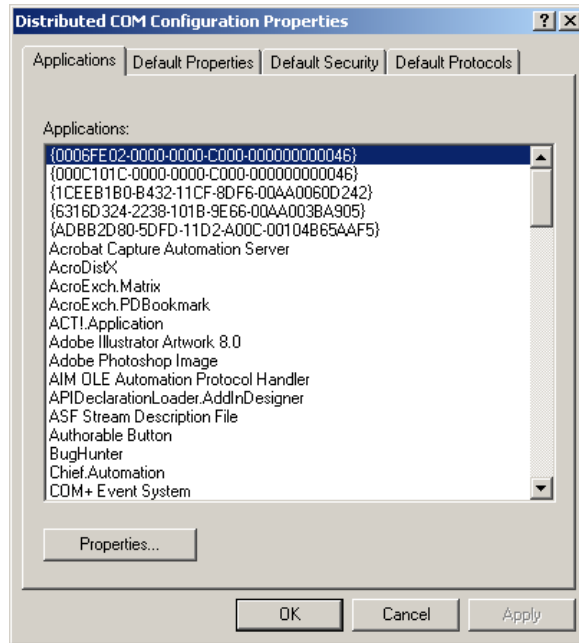
1. Click the Windows **Start** button.
2. Select **Run ...**

*The Run dialog appears.*



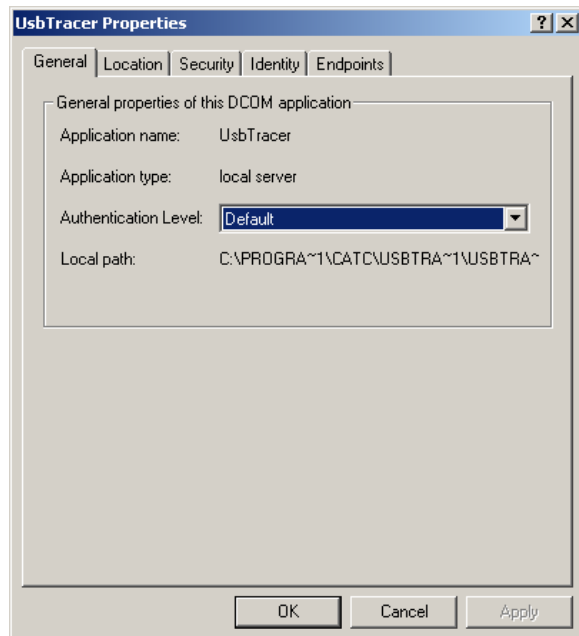
3. In the Open edit box within the Run dialog, type **dcomcnfg**.
4. Click **OK**.

*The DCOM configuration dialog box opens.*



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.*



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

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

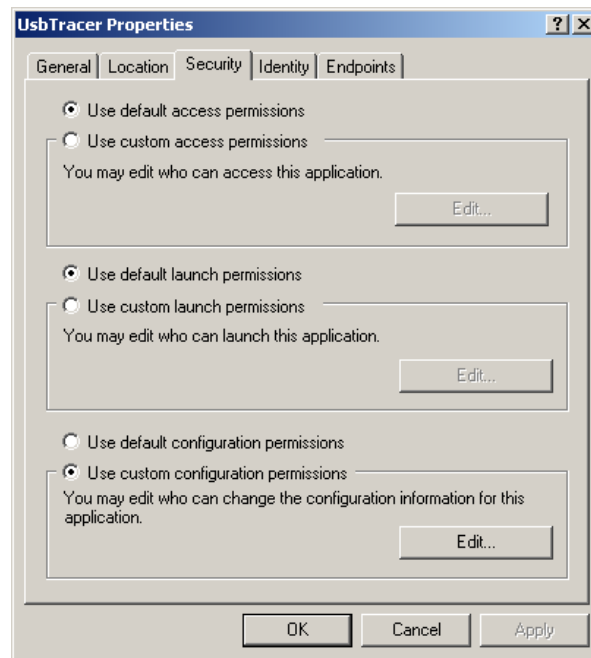


## Configuring Access Permissions

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 displaying, select the **Security** tab.

*The following settings display:*



*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

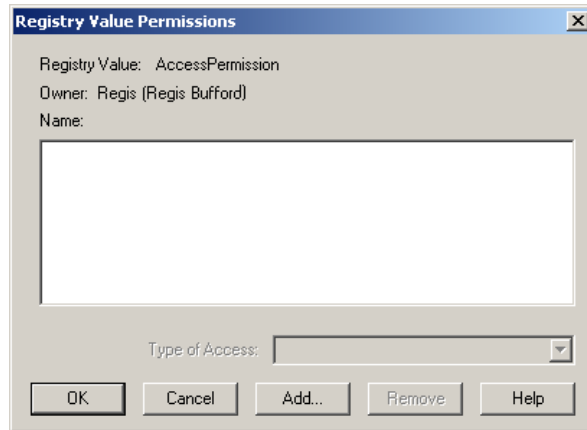
**Configuration Permissions** - Determines who may configure permissions for the application

2. Click the **User Custom Access permissions** option.

*You are going to give all users permission to execute commands on the server.*

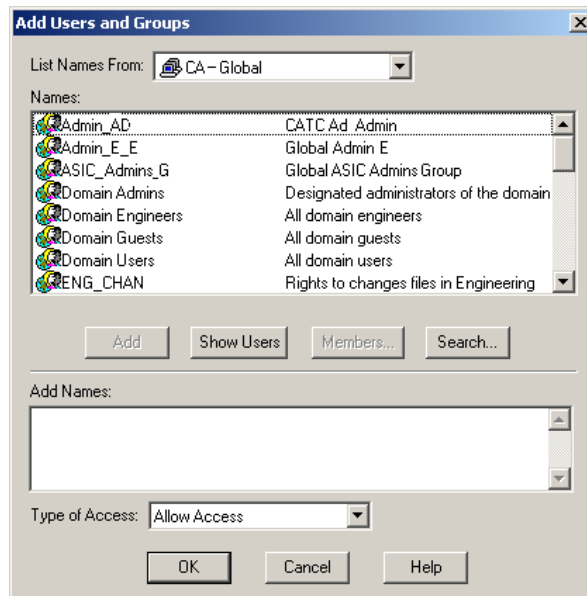
3. Click the **Edit ...** button.

*The Registry Value Permissions dialog box appears.*



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

*The Add Users and Groups dialog box appears.*



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 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**.

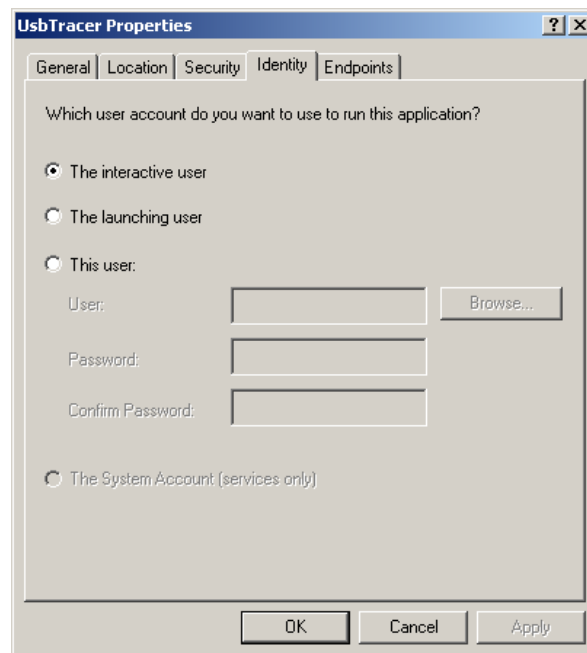
*The Registry Key Permissions dialog box closes. The Properties dialog box remains on the screen.*

## 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:*



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

## A.5. DCOM Client Configuration

DCOM client configuration is a four-step process:

3. Set authentication to occur upon connection with the Host Controller.
4. Set the Host name or IP address of the server that the client will be connecting to.
5. Set commands to run on the Host Controller.
6. Set the network protocol over which DCOM will run.

### Setting When and How Authentication Should Occur

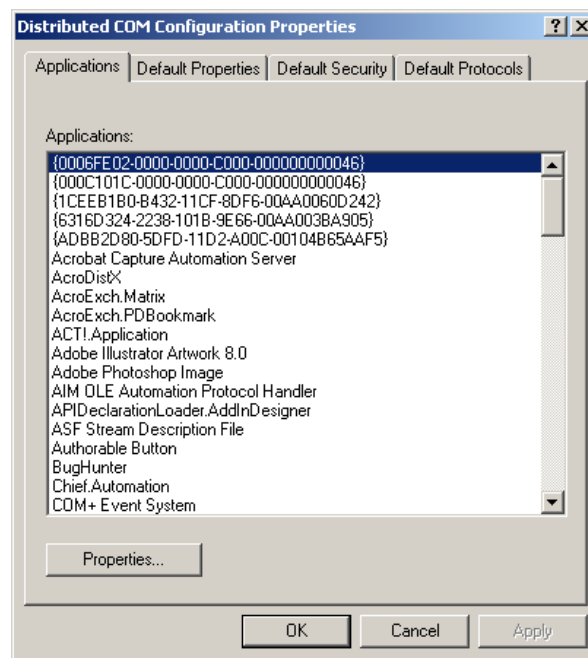
You specify when and how the client should be authenticated by performing the following steps:

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.**

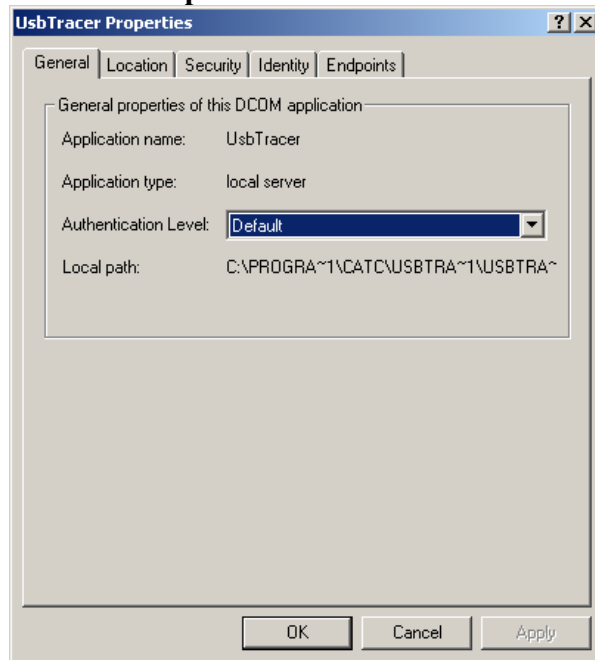
*The following dialog box will appear:*



*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.



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.

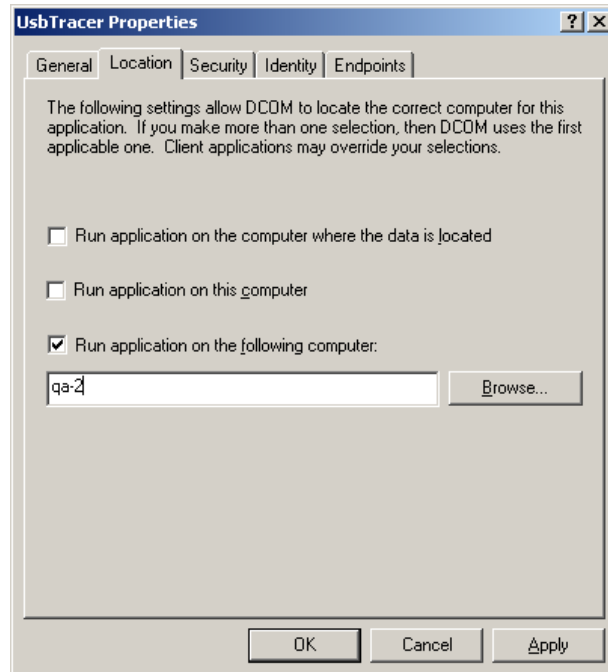
### Set the Host Name or IP Address of the Host Controller

You need to identify the device upon which the client will be executing its commands.

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

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

*The Location window displays.*



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**.

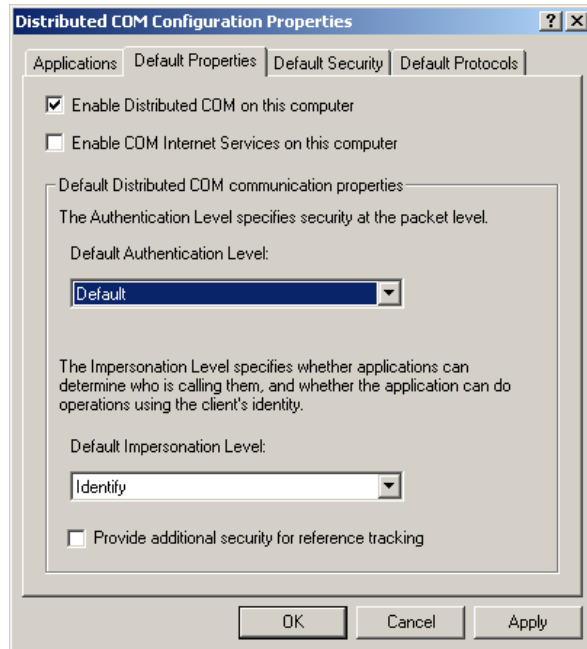
*The Properties dialog box closes. You will see the Distributed COM Configuration Properties dialog box.*

### **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 will tell 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.*



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

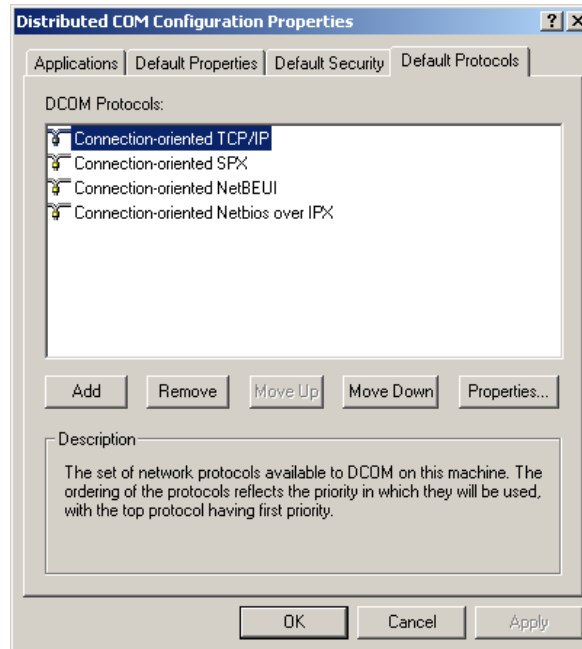
You will need to perform this step on both computers - the DCOM server and DCOM client.

### Set the Default Network Protocol

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

*The following screen will display. 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 will need to add it using the **Add** button.*





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 **analyzer1.html**, or double-click on one of the wsh files. If your setup is correct, execute a CATC sample script on the client PC that will be connecting to the Host Controller. Sample script files can be found on the Automation Installation diskette.

### Testing Your Automation Setup with AnalyzerClient.exe

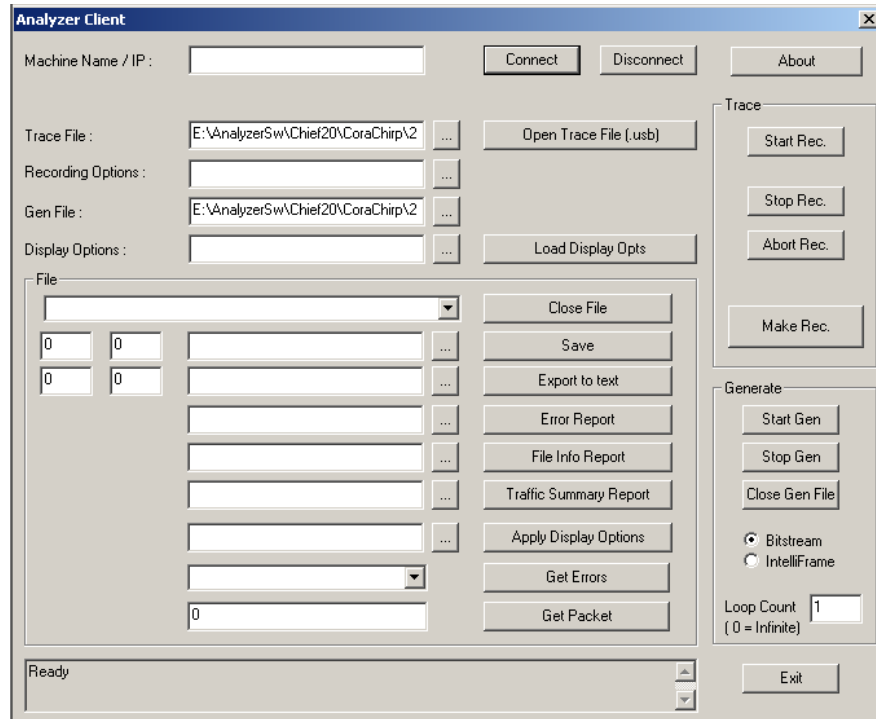
The following steps show you how to test your automation setup using the **AnalyzerClient.exe**. As mentioned above, you will need to compile this first from the source code in the **cpp** directory. Please refer to the ReadMe file in the Automation package for information about compiling the application.

In the steps below, you execute **analyzerclient.exe** on a remote PC and establish a connection with the Host Controller PC. Afterwards, you will 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 will start. If you prefer, you can copy the executable to your hard drive and launch it from there.*




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 will automatically launch 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 will need to 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 CATC

Type of Service	Contract
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 ...	Computer Access Technology Corporation Customer Support 2403 Walsh Avenue Santa Clara, CA 95051-1302
Send e-mail...	<a href="mailto:support@CATC.com">support@CATC.com</a>
Visit CATC's web site...	<a href="http://www.CATC.com/">http://www.CATC.com/</a>

## Warranty and License

Computer Access Technology Corporation (hereafter CATC) warrants this product to be free from defects in material, content, and workmanship, and agrees to repair or replace any of the enclosed unit that proves defective under these terms conditions. Parts and labor are warranted for one year from date of first purchase.



part  
and  
the

The CATC software is licensed for use on a single personal computer. The software may be copied for backup purposes only.

This warranty covers all defects in material or workmanship. It does not cover accidents, misuse, neglect, unauthorized product modification, or acts of nature. Except as expressly provided above, CATC makes no warranties or conditions, express, implied, or statutory, including without limitation the implied warranties of merchantability and fitness for a particular purpose.

CATC shall not be liable for damage to other property caused by any defects in this product, damages based upon inconvenience, loss of use of the product, loss of time or data, commercial loss, or any other damages, whether special, incidental, consequential, or otherwise, whether under theory of contract, tort (including negligence), indemnity, product liability, or otherwise. In no event shall CATC's liability exceed the total amount paid to CATC for this product.

CATC reserves the right to revise these specifications without notice or penalty.