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

Automation API

Version 1.1

Reference Manual

for

LeCroy UWBTracer™

Version 2.0

Manual Version 2.00
March 17, 2006

Document Disclaimer

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

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

Trademarks and Servicemarks

LeCroy, CATC, UWBTracer, UWBTracer MPI, and *UWBTracer Automation* are trademarks of LeCroy.

Microsoft is a registered trademark of Microsoft Inc.

All other trademarks are property of their respective companies.

Copyright

Table of Contents

Table of Figures

1 Introduction

LeCroy UWBTracer™ software provides a rich functional COM/Automation API to the most important functionalities of the LeCroy UWBTracer Protocol Analyzer, which makes it a great tool for implementation of various complicated testing/development/debugging automated programs. The "dual" nature of the interfaces provided makes it easy to use the UWBTracer COM API in different IDE (Integrated Development Environment) supporting COM architecture.

A special support for typeless script languages (like VB or JavaScript) overriding some restrictions imposed by script engines (regarding remote access, dynamic object creation and handling events) gives the opportunity to write client applications very quickly and easily without having significant programming skills or buying/installing expensive and powerful programming language systems. All these features along with the capability to set up all necessary DCOM permissions during the installation process make the LeCroy UWBTracer analyzer a very attractive tool able to automate and speed up many engineering processes.

NOTE: In order to use UWBTracer COM API the application should be registered as a COM server in a system registry. Basically, it is done during the installation process, but you can always reregister it by typing in command line:

uwbtracer – regserver

(You should have permissions to write to the registry, though). If you would like to access UWBTracer application remotely you should install the UWBTracer application on both server and client machine and accept enabling remote access option during the installation.

System Requirements

Automation API version 1.1 was introduced in the UWBTracer MPI version 1.20 software. It is supported in UWBTracer MPI version 2.0 and higher software.

Support Resources

As new functionalities are added to the API, not all of them are supported by older versions of the analyzer software. For the current release of analyzer software, please go to:
http://www.lecroy.com

Setting Up Automation for Local Use

If you intend to run Automation on the analyzer's 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.

Setting Up Automation for Remote Use

If you intend to run automation remotely over a network, you will need to perform DCOM configuration.

2 Automation API Description

LeCroy UwbAnalyzer API exposes the following objects and interfaces:

Objects	Interfaces	Description
`UwbAnalyzer`		
	`IUwbAnalyzer`	Primary analyzer interface
	`_IAnalyzerEvents`	Analyzer event source
`UwbTrace`		
	`IUwbTrace` `IUwbTrace2`	Trace file interfaces

Only `UwbAnalyzer` object is creatable at the top level (that is, via `CoCreateInstance` COM API function or other COM object creation methods), instantiation of objects of other classes requires API calls. The following diagram represents the objects dependencies:

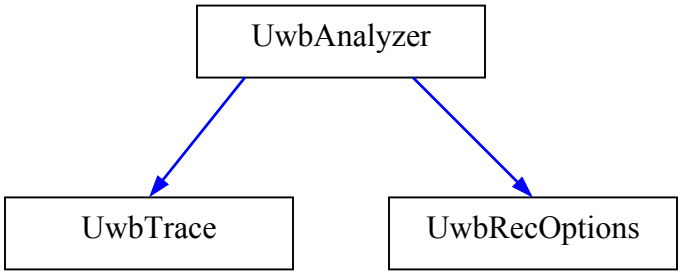

Figure 2-1 API Interfaces

All interfaces are dual interfaces, which allows simple use from typeless languages (i.e. VB/Java Script) as well as from C++.

All objects implement `ISupportErrorInfo` interface allowing easy error handling on the client side.

Some examples of C++ code given in this document assume using "import" (Microsoft® C++ - specific) technique of creating COM clients; that means the corresponding include is used:

```
#include "UwbAutomation.tlb" no_namespace named_guids
```

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

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

NOTE: This reference gives descriptions ONLY for implemented properties and methods of the interfaces.

3 Primary Dual Interface for Analyzer

3.1 *IUwbAnalyzer dual interface*

`IUwbAnalyzer` interface is the primary interface for `UwbAnalyzer` object that gives access to the most important functionalities of LeCroy UWBTracer™ analyzer.

Class ID: **5C7F6A69-7439-4248-852C-45A0A9462D4A**
Prog ID: **CATC.UwbAnalyzer**
COM server: **UWBTracer.exe**
IUwbAnalyzer IID: **0B179BB7-DC61-11d4-9B71-000102566088**

3.1.1 IUwbAnalyzer::get_ApplicationFolder (property)

```
HRESULT get_ApplicationFolder( [out, retval] BSTR *app_folder );
```

Retrieves the full local path to the folder where the UWBTracer application was installed and registered as a COM server.

Parameters

`app_folder` - Pointer to string variable where to put UWBTracer application folder

Return values

Remarks
This property allows client applications to access some important files used by UWBTracer application like recording options, display options, trace files etc. in places relatively to UWBTracer application folder. If sometime later UWBTracer application is re-installed to another location the client code doesn't have to be changed to reflect this event.

Example

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

        ' Open trace file 'some_trace.uwb' in the UWBTracer folder
        Set Trace = Analyzer.OpenFile( Analyzer.ApplicationFolder & "some_trace.uwb" )

        ' Save opened trace as 'some_trace1.uwb' in the UWBTracer folder
        Trace.Save Analyzer.ApplicationFolder & "some_trace1.uwb"
        ...
C++:

        HRESULT  hr;
        IUwbAnalyzer*  poUwbAnalyzer;

        // create UwbAnalyzer object
        if ( FAILED( CoCreateInstance( CLSID_UwbAnalyzer, NULL, CLSCTX_SERVER, IID_IUwbAnalyzer,
                 (LPVOID *)&poUwbAnalyzer ) )
                return;

        BSTR app_folder;
        hr = poUwbAnalyzer->get_ApplicationFolder( &app_folder );

        // . . . Do something with app_folder


        SysFreeString( app_folder ); // Free BSTR resources
        poUwbAnalyzer->Release();    // Release analyzer object
```

3.1.2 IUwbAnalyzer::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.UwbAnalyzer")
        SwVersion = Analyzer.GetVersion(0)
        BEVersion = Analyzer.GetVersion(1)
        FwVersion = Analyzer.GetVersion(2)
        MsgBox "Software:" & CStr(SwVersion) & ", BusEngine:" & CStr(BEVersion) & ", Firmware:" &
               CStr(FwVersion)

C++:

        HRESULT   hr;
        IUwbAnalyzer*  poUwbAnalyzer;

        // create UwbAnalyzer object
        if ( FAILED( CoCreateInstance(
                 CLSID_UwbAnalyzer,
                 NULL, CLSCTX_SERVER,
                 IID_IUwbAnalyzer,
                 (LPVOID *)&poUwbAnalyzer ) )
                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("UwbAnalyzer client"), MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"), MB_OK );
                return 1;
        }

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

3.1.3 IUwbAnalyzer::GetSerialNumber

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

Retrieves serial number of analyzer device

Parameters

Return values

aANALYZERCOMERROR_INVALIDVERSIONTYPE - specified version type is invalid
aANALYZERCOMERROR_ANALYZERNOTCONNECTED - analyzer device is not connected

Remarks

Example

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


C++:

        HRESULT   hr;
        IUwbAnalyzer*  poUwbAnalyzer;

        // create UwbAnalyzer object
        if ( FAILED( CoCreateInstance(
                 CLSID_UwbAnalyzer,
                 NULL, CLSCTX_SERVER,
                 IID_IUwbAnalyzer,
                 (LPVOID *)&poUwbAnalyzer ) )
                return;

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

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

3.1.4 IUwbAnalyzer::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 `UwbTrace` object primary interface

Return values

`ANALYZERCOMERROR_UNABLEOPENFILE` – unable to open file

Remarks

`UwbTrace` 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.UwbAnalyzer")
Set Trace = Analyzer.OpenFile (CurrentDir & "your_trace_file.uwb")
```

C++:

```
HRESULT    hr;
IUwbAnalyzer*  poUwbAnalyzer;

// create UwbAnalyzer object
if ( FAILED( CoCreateInstance(
         CLSID_UwbAnalyzer,
         NULL, CLSCTX_SERVER,
         IID_IUwbAnalyzer,
         (LPVOID *)&poUwbAnalyzer ) )
        return;

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

// query for IUwbTrace interface
IUwbTrace* UWB_trace;
hr = trace->QueryInterface( IID_IUwbTrace, (LPVOID *)&UWB_trace );
trace->Release();

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

3.1.5 IUwbAnalyzer::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

`ANALYZERCOMERROR_UNABLESTARTRECORDING` - unable to start recording

Remarks

After recording starts this function will return. Analyzer continues recording until it is finished or until either "`StopRecording`" or "`StopRecordingAndWaitForTrace`" 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 UWBTracer application. You can create such file when you select "Setup – Recording Options…" from UWBTracer application menu, change the recording options in the appeared dialog and select "Save…" button.

Example

VBScript:

```
<OBJECT
        RUNAT=Server
        ID = Analyzer
        CLASSID = "clsid:7A4ECA40-E668-11D4-9B7C-000102566088"
>
</OBJECT>

...<INPUT TYPE=TEXT   VALUE="" NAME="TextRecOptions"> ...
...<INPUT TYPE=BUTTON VALUE="" NAME="BtnStartRecording"> ...

<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++:

```
IUwbAnalyzer* uwb_analyzer;
BSTR          ro_file_name;

. . .

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

3.1.6 IUwbAnalyzer::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

`ANALYZERCOMERROR_UNABLESTOPRECORDING` - error stopping recording

Remarks
Stops recording started by 'StartRecording' method. The event will be 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:EE43DE29-65CA-4904-B111-32919E00D061"
	>
	</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++:

```
	IUwbAnalyzer* UWB_analyzer;

	. . .

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

3.1.7 IUwbAnalyzer::StopRecordingAndWaitForTrace

```
HRESULT StopRecordingAndWaitForTrace
          ( [out, retval] IDispatch** trace );
```

Stops recording started by `StartRecording` method and returns pointer to the trace object created.

Parameters

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

Return values

`ANALYZERCOMERROR_UNABLESTOPRECORDING` - error stopping recording

Remarks

This method stops recording started by `'StartRecording'` method and doesn't return until the recording is uploaded and a trace object related to this recording is created. Versus to the 'StopRecording' no event will be issued when recording is stopped.

Example

```
WSH:

Set Analyzer = WScript.CreateObject("CATC.UWBAnalyzer")
CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))

'
' In the piece of code below we perform 4 sequential recordings and
' save the traces recorded in the current folder
'

For I = 1 To 4

  'Tell the LeCroy UWB analyzer to start recording using 'my.rec' recording options file in the
       current folder.
  Analyzer.StartRecording ( CurrentDir & "my.rec" )

  ' Imitation of some activity - just sleep for 3 seconds.
  ' Some real traffic generation code might be put here...
  WScript.Sleep 3000

  ' Tell the analyzer to stop recording and give the trace acquired.
  Set Trace = Analyzer.StopRecordingAndWaitForTrace

  ' Save the trace in the current folder
  Trace.Save CurrentDir & "uwb_seqrec_data" & CStr(I) & ".uwb"

Next

'Release the analyzer ...
Set Analyzer = Nothing
```

C++: (Raw code without using type library. All error handling is omitted for simplicity)

```
        IUwbAnalyzer*  pAnalyzer = NULL;

        // create UwbAnalyzer object
        if ( FAILED( CoCreateInstance( CLSID_UwbAnalyzer, NULL, CLSCTX_SERVER, IID_IUwbAnalyzer,
                 (LPVOID *)&pAnalyzer ) )
                return;

        BSTR ro_file_name = SysAllocString( L"test_ro.rec" );
        BSTR bstr_trace_name;

        IDispatch* trace = NULL;

        for( int i = 0; i < 4; i++ )
        {
                if( FAILED( pAnalyzer ->StartRecording( ro_file_name ) )
                        return;  // error handling and resources releasing should be done

               // Fake generation...
               Sleep(3000);

               if( FAILED( Analyzer->StopRecordingAndWaitForTrace( &trace ) ) )
                        return;  // error handling and resources releasing  should be done


                IUwbTrace* uwbTrace = NULL;

                // Query for IUwbTrace interface
                if( FAILED( trace->QueryInterface( IID_IUwbTrace, (LPVOID*)uwbTrace ) ) )
                               return; // error handling and resources releasing should be done


               OLECHAR trace_name[_MAX_PATH];
               swprintf ( trace_name, "test_data%u.uwb", i );
               SysAllocString( bstr_trace_name );


               // Save the trace recorded to the current folder
               uwbTrace->SaveCopyTo( bstr_trace_name );

               // release the trace object
               trace->Release();
               uwbTrace->Release();

               SysFreeString( bstr_trace_name ); // free BSTR
         }

        pAnalyzer->Release();          // release the analyzer object
        SysFreeString( ro_file_name ); // free BSTR
```

C++: (Microsoft® C++ specific, using type library. All error handling is omitted for simplicity.)

```
IUwbAnalyzerPtr   pAnalyzer = NULL;
pAnalyzer.CreateInstance( __uuidof(UwbAnalyzer) );

if( !pAnalyzer ) return;

IUwbTracePtr trace = NULL;

for( int i = 0; i < 4; i++ )
{
        // Start recording using recording options file 'test_ro.rec' located in the
        current folder
        pAnalyzer->StartRecording( _bstr_t("test_ro.rec") );

        // Fake generation... just sleep for 3 second
        Sleep(3000);

        trace = pAnalyzer->StopRecordingAndWaitForTrace();


        TCHAR trace_name[_MAX_PATH];
        sprintf( trace_name, "test_data%u.uwb", i );

        // Save the trace recorded in the current folder
        trace->SaveAs( _bstr_t(trace_name) );


        // Release the trace object
        trace = NULL;
 }
```

3.1.8 IUwbAnalyzer::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 `UwbTrace` object primary interface

Return values

Remarks
This method acts like `'StartRecording'` method but will not return until recording is completed.

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

Recording options file is the file with extension .rec created by UwbAnalyzer application. You can create such file when you select “Setup – Recording Options…” from UwbAnalyzer 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.UwbAnalyzer")
Set Trace = Analyzer.MakeRecording (CurrentDir & "Input\test_ro.rec")
```

C++:

```
IDispatch*    trace;
IUwbAnalyzer* UWB_analyzer;
BSTR          ro_file_name;
HRESULT       hr;

. . .
try
{
      trace = UWB_analyzer->MakeRecording( ro_file_name )
}
catch ( _com_error& er)
{
      if (er.Description().length() > 0)
            ::MessageBox( NULL, er.Description(),  _T("UwbAnalyzer client"), MB_OK );
      else
            ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"), MB_OK );
      return 1;
}
// query for VTBL interface
IUwbTrace* UWB_trace;
hr = trace->QueryInterface( IID_IUwbTrace, (LPVOID *)&UWB_trace );
trace->Release();
```

3.1.9 IUwbAnalyzer::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 UwbAnalyzer application. You can create such file when you select “Setup – Display Options…” from UwbAnalyzer application menu, change the display options in the appeared dialog and select “Save…” button.

Example

See `IUwbTrace::ApplyDisplayOptions`

4 Primary Dual Interface for Trace

4.1 *IUwbTrace dual interface*

`IUwbTrace` interface is the primary interface for `UwbTrace` object.

IID: 0B179BB9-DC61-11D4-9B71-000102566088

4.1.1 IUwbTrace::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.UwbAnalyzer")
        CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
        Analyzer.StartRecording (CurrentDir & "test_ro.rec")
        '... do something
        Set Trace = Analyzer.StopRecordingAndWaitForTrace
        MsgBox "Trace name " & Trace.GetName

C++:
        IUwbTrace* UWB_trace;

        . . .

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

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

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

4.1.2 IUwbTrace::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 UwbAnalyzer application. You can create such file when you select “Setup – Display Options…” from UwbAnalyzer application menu, change the display options in the appeared dialog and select “Save…” button.

Example

```
WSH:
       Set Analyzer = WScript.CreateObject("CATC.UwbAnalyzer")
       CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
       Analyzer.StartRecording (CurrentDir & "test_ro.rec")
       '... do something
       Set Trace = Analyzer.StopRecordingAndWaitForTrace
       Trace.ApplyDisplayOptions    CurrentDir & "Input\test_do.opt"
       Trace.Save                   CurrentDir & "Output\saved_file.uwb"

C++:

       IUwbTrace* UWB_trace;
       TCHAR file_name[_MAX_PATH];

       . . .

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

4.1.3 IUwbTrace::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 `IUwbTrace::ApplyDisplayOptions`, `IUwbAnalyzer::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.UwbAnalyzer")
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.uwb"
```

C++:

```
IUwbTrace* UWB_trace;
TCHAR file_name[_MAX_PATH];
LONG packet_from;
LONG packet_to;

. . .

try
{
        UWB_trace->Save( file_name, packet_from, packet_to );
}
catch ( _com_error& er)
{
        if (er.Description().length() > 0)
                ::MessageBox( NULL, er.Description(),  _T("UwbAnalyzer client"), MB_OK );
        else
                ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"), MB_OK );
        return 1;
}
```

4.1.4 IUwbTrace::SaveAs

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

Saves trace into a file and makes the current IUwbTrace pointer to refer to this file.

Parameters

`file_name` - string providing the full pathname to file where trace is saved

Return values

`ANALYZERCOMERROR_UNABLESAVE` - unable to save trace file

Remarks

Use this method if you want to move recorded or opened trace into the file.

Example

```
WSH:
	Set Analyzer = WScript.CreateObject("CATC.UwbAnalyzer")
	CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
	Analyzer.StartRecording (CurrentDir & "test_ro.rec")

	'... do something

	Set Trace = Analyzer.StopRecordingAndWaitForTrace
	Trace.SaveAs CurrentDir & "Output\saved_file.uwb"
C++:

	IUwbTrace* UWB_trace;
	TCHAR file_name[_MAX_PATH];
	LONG packet_from;
	LONG packet_to;

	...

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

4.1.5 IUwbTrace::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 `IUwbTrace::ApplyDisplayOptions`, `IUwbAnalyzer::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:

```
File C:\UWB Traces\RealTrace.uwb.
From Frame #0 to Frame #3.

Frame#
_______|_____________________________________________________________________
Frame(0) PHY MAC Control Dest ID(0x0088) Src ID(0xBEEF) Ctl
_______| Data(36 bytes) Rpt Frm Duration(8.853 µs) PHY_ACT( 13.331 µs)
_______| Delta Time(171.125 µs) Idle(157.794 µs) Time Stamp(2.370813167)
_______|_____________________________________________________________________
Frame(1) PHY MAC Data Dest ID(0x0000) Src ID(0xBEEF) Data(258 bytes) Rpt
_______| Frm Duration( 42.153 µs) PHY_ACT( 47.076 µs) Delta Time( 40.114 ms)
_______| Idle( 40.067 ms) Time Stamp(2.370984292)
_______|_____________________________________________________________________
Frame(2) PHY MAC Control Dest ID(0x0088) Src ID(0xBEEF) Ctl
_______| Data(36 bytes) Rpt Frm Duration(8.853 µs) PHY_ACT( 13.316 µs)
_______| Delta Time(165.185 µs) Idle(151.869 µs) Time Stamp(2.411098512)
_______|_____________________________________________________________________
Frame(3) PHY MAC Data Dest ID(0x0000) Src ID(0xBEEF) Data(258 bytes) Rpt
_______| Frm Duration( 42.153 µs) PHY_ACT( 47.076 µs) Delta Time( 39.916 ms)
_______| Idle( 39.869 ms) Time Stamp(2.411263697)
_______|_____________________________________________________________________
```

Example

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

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

4.1.6 IUwbTrace::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 : management.uwb
Trace occurred : Wednesday, February 26, 2003  11:19:51

Number of frames: 3186
Trigger frame number: 0

Recorded with 'LeCroy UWBTracer' analyzer, version 0.90 ( Build 2 )
Analyzer Serial Number: 00111

Motherboard: 0x1 Version: 0x1
   Firmware version: 0.90 ( ROM 1.00 )
   BusEngine version: 0.90
   BusEngine type: 0
   UPAS Slot 1 -  Part Number: ,   PlugIn ID: 0x22,   Version: 0x2
   UPAS Slot 2 -  Part Number: ,   PlugIn ID: 0x22,   Version: 0x2
Number of markers : 1

Recording Options :
  Options Name : Default
  Recording Mode : Snapshot

  Buffer Size : 1.600 MB
  Post-trigger position : 50%
  Base filename & path : D:\Projects\UWBTracer\Debug\data.uwb
```

Example

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

C++:
        IUwbTrace* UWB_trace;
        TCHAR file_name[_MAX_PATH];
        ...
        try
        {
                UWB_trace->ReportFileInfo( file_name );
        }
        catch ( _com_error& er)
        {
                if (er.Description().length() > 0)
                        ::MessageBox( NULL, er.Description(),  _T("UwbAnalyzer client"), MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"), MB_OK );
                return 1;
        }
```

4.1.7 IUwbTrace::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.UwbAnalyzer")
        CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
        Analyzer.StartRecording (CurrentDir & "test_ro.rec")

        '... do something

        Set Trace = Analyzer.StopRecordingAndWaitForTrace
        MsgBox Trace.GetPacketsCount & " packets recorded"

C++:
        IUwbTrace* UWB_trace;

        . . .

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

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

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

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

4.1.8 IUwbTrace::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.UwbAnalyzer")
       Analyzer.StartRecording (CurrentDir & "test_ro.rec")

       '... do something

       Set Trace = Analyzer.StopRecordingAndWaitForTrace
       TriggerPacket = Trace.GetTriggerPacketNum
       Trace.Save CurrentDir & "trigger_portion.uwb", CInt(TriggerPacket)-5,
CInt(TriggerPacket)+5
       Trace.ExportToText CurrentDir & "trigger_portion.txt", CInt(TriggerPacket)-5,
CInt(ErrorPacket)+5

C++:
       See an example for IUwbTrace::GetPacketsCount
```

4.2 *IUwbTrace2 interface*

`IUwbTrace2` interface is the second interface for the `UwbTrace` object. It inherits and extends some trace-related functionality exposed via the `IUwbTrace` interface.

IID : E69C91C8-F729-4675-92C3-46C7AD071BEE

4.2.1 IUwbTrace2::GotoTime

```
HRESULT GotoTime ( [in] double time );
```

Instructs the trace view to jump to the specified timestamp.

Parameters

`time` – the time (in nanoseconds) to jump

Example

```
WSH:
        Set Analyzer = WScript.CreateObject("CATC.UwbAnalyzer")
        CurrentDir = Left(WScript.ScriptFullName, InstrRev(WScript.ScriptFullName, "\"))
        Analyzer.StartRecording (CurrentDir & "test_ro.rec")
        '... do something
        Set Trace = Analyzer.StopRecordingAndWaitForTrace

        Trace.GotoTime( 1000000 ) ' goto 1 millisecond timestap

C++:
        IUwbTrace2* Uwb_trace;
        . . .


        try
        {
                double t = 1000000.0;
                Uwb_trace ->GotoTime( t );
        }
        catch ( _com_error& er)
        {
                if (er.Description().length() > 0)
                        ::MessageBox( NULL, er.Description(),  _T("UwbAnalyzer client"), MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"), MB_OK );
                return 1;
        }
```

4.2.2 IUwbTrace2::ExportToCsv

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

Exports a trace into a text file in CSV format and allows exporting a range of packets.

Parameters

`file_name` - string providing the full pathname of the trace export file

`packet_from` - beginning packet number when you are exporting a range of packets. A value of –1 makes the first packet of the exported trace be the first packet of the trace.

`packet_to` - ending packet number when you are exporting a range of packets. A value of –1 makes the last packet of the exported trace be the last packet of the trace.

Return values

`ANALYZERCOMERROR_UNABLESAVE` – unable to export trace file

Remarks

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

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

5 Analyzer Events Callback Interface

5.1 *_IAnalyzerEvents dispinterface*

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

Since this interface is default source interface for `UwbAnalyzer` 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 `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()
. . .
}
```

Note that due to the bug Q237771 in Microsoft® Active Template Library you need to implement the GetFuncInfoFromId method:

```
        HRESULT GetFuncInfoFromId(const IID& iid, DISPID dispidMember,LCID lcid, _ATL_FUNC_INFO&
info)
        {
                HRESULT hr = IDispEventImpl<IDC_SRCOBJ, CAnalyzerSink>::GetFuncInfoFromId(
                                iid, dispidMember, lcid, info);
                if (SUCCEEDED(hr))
                {
                        if (info.pVarTypes[dispidMember-1] == VT_USERDEFINED)
                                info.pVarTypes[dispidMember-1] = VT_I4;
                }
                return hr;
        }
```

This implementation allows you to use `EAnalyzerState` enumerator in the callback interface.

Then, after you have established the connection with the server, you need to advise your implementation of the event interface:

```
hr = CoCreateInstance( CLSID_UwbAnalyzer, NULL,
        CLSCTX_SERVER, IID_IUwbAnalyzer, (LPVOID *)&m_poUwbAnalyzer );

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_poUwbAnalyzer, &m_poAnalyzerSink->m_lFCid,
                      &m_poAnalyzerSink->m_iid, &m_poAnalyzerSink->m_wMajorVerNum,
                      &m_poAnalyzerSink->m_wMinorVerNum);

 if ( FAILED(hr) )
        return 1;

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

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

5.1.1 _IAnalyzerEvents::OnTraceCreated

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

Fired when trace is created; this event is a result of IUwbAnalyzer::StartRecording/ IUwbAnalyzer::StopRecording method call

Parameters

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

Return values

Remarks

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

Example

VBScript:

```
<OBJECT
        ID = Analyzer
        CLASSID = "clsid:EE43DE29-65CA-4904-B111-32919E00D061" >
</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 )
{
        IUwbTrace* UWB_trace;
        HRESULT hr;
        hr = trace->QueryInterface( IID_IUwbTrace, (void**)&UWB_trace );

        if (FAILED(hr))
        {
                _com_error er(hr);
                if (er.Description().length() > 0)
                        ::MessageBox( NULL, er.Description(),  _T("UwbAnalyzer client"),
                MB_OK );
                else
                        ::MessageBox( NULL, er.ErrorMessage(), _T("UwbAnalyzer client"),
                MB_OK );
                return hr;
         }

        . . .

        return hr;
}
```

5.1.2 _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` - not used,

`state` - current analyzer state; `EanalyzerState` enumerator has the following values:

`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. When the analyzer state is `ANALYZERSTATE_IDLE`, this parameter is not applicable.

When the analyzer state is ANALYZERSTATE_WAITING_TRIGGER or ANALYZERSTATE_RECORDING_TRIGGERED, this parameter shows analyzer memory utilization.

When the analyzer state is `ANALYZERSTATE_UPLOADING_DATA`, this parameter shows the percent of data uploaded.

When the 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:EE43DE29-65CA-4904-B111-32919E00D061" >
</OBJECT>
<P ALIGN=LEFT ID=StatusText></P>

<SCRIPT LANGUAGE="VBScript">
```

```
        <!--
        Sub Analyzer_OnStatusReport(ByVal System, ByVal State, ByVal Percent)
                Select Case State
                        Case -1
                                StatusText.innerText = "Idle"
                        Case 0
                                StatusText.innerText = "Recording - Waiting for trigger, " &
        Percent & "% done"
                        Case 1
                                StatusText.innerText = "Recording - Triggered, " & Percent & "%
        done"
                        Case 2
                                StatusText.innerText = "Uploading data, " & Percent & "% done"
                        Case 3
                                StatusText.innerText = "Saving data, " & Percent & "% done"
                        Case Else
                                StatusText.innerText = "State unknown, " & Percent & "% done"
                End Select
        End Sub
        -->
        </SCRIPT>
```

C++:

```
        #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 )
        {

                TCHAR buf[1024];
                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;
                }

                if ( ANALYZERSTATE_IDLE != state )
                        _stprintf( buf, _T("%s, done %ld%%"), status_buf, percent_done );
                else
                        _stprintf( buf, _T("%s"), status_buf );

                ::SetWindowText( m_hwndStatus, buf );

                return S_OK;
        }
```

6 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 UWBTracer™ 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 UWBTracer analyzer server to create automation objects dynamically, handle events, and control operation remotely.

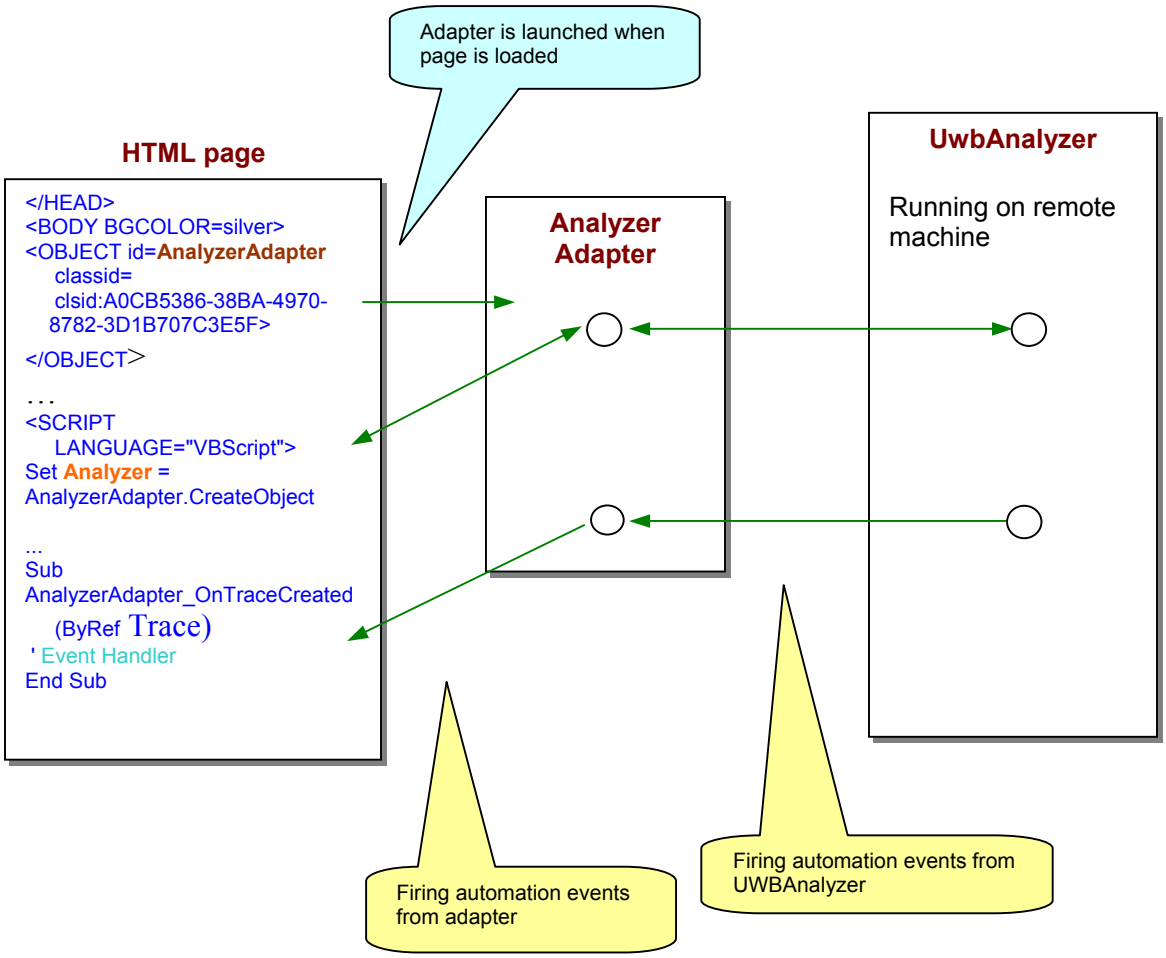

Figure 6-1 Analyzer Adapter

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

Primary default interface:  IanalazerAdapter
```

6.1 *IanalyzerAdapter Interface*

6.1.1 IAnalyzerAdapter::CreateObject

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

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

Parameters

`class_id` - string representation of classid, or ProgId ("clsid:EE43DE29-65CA-4904-B111-32919E00D061" or "CATC.UWBAnalyzer" for LeCroy UWB 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 LeCroy analyzer COM servers can be instantiated through this method. The method Detach (see below) should be called when the work with the remote object is completed. (NOTE: The pointer returned in `ppNewObj` should be released separately.)

Example

VBScript:

```
        </HEAD>
        <OBJECT id=AnalyzerAdapter
                classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
        </OBJECT>
        ...
        <input type="button" value="Connect" name="BtnConnect">
        <INPUT NAME="RemoteServer">

        <SCRIPT LANGUAGE="VBScript">
        <!--
        Sub BtnConnect_onclick
                On Error Resume Next

                Set Analyzer = AnalyzerAdapter.CreateObject("CATC.UWBAnalyzer", RemoteServer.value
        )

                if Not Analyzer Is Nothing Then
                        window.status = "UWBTracer connected"
                else
                        msg = "Unable to connect to UWBTracer"
                        MsgBox msg, vbCritical
                        window.status = msg
                End If
        End Sub
        -->
        </SCRIPT>
```

WSH:

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

        RemoteServer = "EVEREST"
        Set Analyzer = AnalyzerAdapter.CreateObject("CATC.UWBAnalyzer", RemoteServer)

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

6.1.2 IAnalyzerAdapter::Attach

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

This method attaches LeCroy analyzer object to the adapter.

Parameters

`pObj` – pointer to the LeCroy analyzer object to be attached.

Return values

Remarks

Only LeCroy analyzer COM servers can be attached to the adapter. If some other analyzer object was previously attached to the adapter it will be 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 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.UWBAnalyzer" ) 'VBScript function creates object
        locally

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

                        window.status = "UWBTracer connected"
                else
                        msg = "Unable to connect to UWBTracer"
                        MsgBox msg, vbCritical
                        window.status = msg
                End If
        End Sub

        -->
        </SCRIPT>
```

WSH:

```
        ' Create LeCroy 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" )
        ...
```

6.1.3 IAnalyzerAdapter::Detach

```
HRESULT Detach();
```

This method detaches the LeCroy analyzer object from the adapter.

Parameters

Return values

Remarks

This method detaches an analyzer object from the adapter. This method doesn't guarantee that all resources associated with the detached object will be 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.UWBAnalyzer",
        RemoteServer.value)

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

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

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

6.1.4 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 LeCroy analyzer COM servers can be attached to the adapter.

Example

VBScript:

```
</HEAD>
<OBJECT id=AnalyzerAdapter
        classid=clsid:A0CB5386-38BA-4970-8782-3D1B707C3E5F>
</OBJECT>
...
<input type="button" value="Connect"    name="BtnConnect">
<input type="button" value="Disconnect" name="BtnDisconnect">
<INPUT NAME="RemoteServer">

<SCRIPT LANGUAGE="VBScript">
<!--
Sub BtnConnect_onclick

       'Launch MS Excel instead of UWBTracer !!!
       Set Analyzer = CreateObject("Excel.Application")
       Analyzer.Visible = True

       If Not AnalyzerAdapter.IsValidObject( Analyzer ) Then
              MsgBox "The object cannot be attached", vbCritical
              Set Analyzer = Nothing
              Exit Sub
       End If
End Sub

-->
</SCRIPT>
```

How to Contact LeCroy

Type of Service	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 …	LeCroy Protocol Solutions Group Customer Support 3385 Scott Blvd. Santa Clara, CA 95054-3115
Send e-mail…	support@catc.com
Visit LeCroy's web site…	http://www.lecroy.com/