

# **Multi-Instrument Automation Server Interfaces**

**Version 1.1**

*Note: VIRTINS TECHNOLOGY reserves the right to make modifications to this document at any time without notice. This document may contain typographical errors.*

## TABLE OF CONTENTS

<b>1. USING MULTI-INSTRUMENT AS AN AUTOMATION SERVER.....</b>	<b>3</b>
<b>2. MULTI-INSTRUMENT AUTOMATION SERVER INTERFACE SPECIFICATIONS.....</b>	<b>3</b>
2.1 SHOWWINDOW .....	3
2.2 OPENWINDOW .....	4
2.3 CLOSEWINDOW .....	5
2.4 MOVEWINDOW.....	5
2.5 TILEWINDOW .....	6
2.6 SHOWMENU .....	6
2.7 HIDE MENU .....	6
2.8 SHOWTOOLBAR .....	7
2.9 HIDE TOOLBAR.....	7
2.10 SHOWTITLEBAR.....	7
2.11 HIDE TITLEBAR .....	8
2.12 SETSIGNALGENERATORPARAMETERS .....	8
2.13 UPDATEMULTITONEITEM .....	13
2.14 SETNUMBEROFMULTITONEITEMS .....	14
2.15 LOADWFLIBRARY.....	14
2.16 STARTDAO .....	15
2.17 STOPDAO.....	15
2.18 GETDAOSTATUS.....	15
2.19 SETDAQPARAMETERS .....	15
2.20 SETVIEWPARAMETERS .....	18
2.21 STARTDAQ .....	20
2.22 STOPDAQ.....	21
2.23 RECORD .....	21
2.24 GETDAQSTATUS.....	21
2.25 GETDDP.....	21
2.26 LOADPANELSETTINGFILE.....	22
2.27 LOADFILE.....	22
2.28 SAVEFILE .....	22
2.29 OSCILLOSCOPEEXPORT.....	22
2.30 SPECTRUMANALYZEREXPORT.....	23
2.31 LOCKGUI.....	23
2.32 UNLOCKGUI .....	23
2.33 UNLOCK.....	23
<b>3. MULTI-INSTRUMENT AUTOMATION CLIENT PROGRAM DEVELOPMENT GUIDE.....</b>	<b>24</b>
3.1 BASIC FILES OF MULTI-INSTRUMENT AUTOMATION SERVER .....	24
3.2 MULTI-INSTRUMENT AUTOMATION SERVER REGISTRATION .....	25
3.3 MULTI-INSTRUMENT AUTOMATION SERVER CLSID AND PROGID .....	25
<b>4. SAMPLE AUTOMATION CLIENT PROGRAMS.....</b>	<b>26</b>
4.1 TESTAUTOMATION WRITTEN IN VISUAL BASIC 6.0 .....	26
4.2 TESTAUTOMATION WRITTEN IN VISUAL C++ 6.0 .....	27

## 1. Using Multi-Instrument as an Automation Server

ActiveX is the general name for a set of Microsoft technologies that allows you to reuse code and link individual programs together to suit your computing needs. Based on COM (Component Object Model) technologies, ActiveX is an extension of a previous technology called OLE (Object Linking and Embedding). Each program does not need to regenerate components, but rather, reuse components to give you the power to combine applications together. ActiveX Automation is a subset of the Microsoft's ActiveX technologies. ActiveX Automation applications interact in a Client/Server model, where the Automation Server exposes objects which can be controlled by the Automation Client (also known as the Automation Controller).

Multi-Instrument offers support for ActiveX Automation as a server, i.e. ActiveX Automation Server (also known as OLE Automation Server, or simply Automation Server). An Automation Client program can control and share the data with the Multi-Instrument Automation Server through the interfaces that the Multi-Instrument Automation Server exposes. In this way, Multi-Instrument can be seamlessly integrated into other applications as a reusable binary component.

In the following chapters, the Multi-Instrument Automation Server Interfaces will be described and the sample Automation Client programs in Visual Basic 6.0 and Visual C++ 6.0 will be provided.

## 2. Multi-Instrument Automation Server Interface Specifications

### 2.1 ShowWindow

The ShowWindow function sets the specified window's show state.

```
void ShowWindow(  
long nWindowID, // Window ID  
long nCmdShow   // show state of window  
);
```

#### Parameters

*nWindowID*

Window ID. It specifies which window to operate.

- 0: Mainframe
- 1: Oscilloscope
- 2: Spectrum Analyzer
- 3: Multimeter
- 4: Generator
- 5: Spectrum 3D Plot
- 60~67: Data Logger
- 90~105: DDP Viewer

*nCmdShow*

Specifies how the window is to be shown. It must be one of the following values:

- 0: SW\_HIDE, hides this window and passes activation to another window;
- 1: SW\_SHOWNORMAL, activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position;
- 2: SW\_SHOWMINIMIZED, activates the window and displays it as an icon;
- 3: SW\_SHOWMAXIMIZED, activates the window and displays it as a maximized window;
- 4: SW\_SHOWNOACTIVATE, displays the window in its most recent size and position. The window that is currently active remains active;
- 5: SW\_SHOW, activates the window and displays it in its current size and position;
- 6: SW\_MINIMIZE, minimizes the window and activates the top-level window in the system's list;
- 7: SW\_SHOWMINNOACTIVE, displays the window as an icon. The window that is currently active remains active;
- 8: SW\_SHOWNA, displays the window in its current state. The window that is currently active remains active;
- 9: SW\_RESTORE, activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position.
- 100: Set the "Show Edit" flag, applicable only if *nWindowID* = 4 (signal generator)
- 101: Reset the "Show Edit" flag, applicable only if *nWindowID* = 4 (signal generator)

### **Remarks**

Except the mainframe window, which is opened automatically once the Multi-Instrument Automation Server is launched and closed once the Multi-Instrument Automation Server is shut down, the rest of windows in Multi-Instrument must be in open state while ShowWindow is called. These windows can be opened using OpenWindow function and closed using CloseWindow function.

## **2.2 OpenWindow**

The OpenWindow function opens the specified window.

```
void OpenWindow(  
long nWindowID, // Window ID  
);
```

### **Parameters**

*nWindowID*

Window ID. It specifies which window to operate.

- 1: Oscilloscope
- 2: Spectrum Analyzer
- 3: Multimeter
- 4: Generator
- 5: Spectrum 3D Plot
- 60~67: Data Logger

90~105: DDP Viewer

## 2.3 CloseWindow

The CloseWindow function closes the specified window.

```
void CloseWindow(  
long nWindowID, // Window ID  
);
```

### Parameters

*nWindowID*

Window ID. It specifies which window to operate.

- 1: Oscilloscope
- 2: Spectrum Analyzer
- 3: Multimeter
- 4: Generator
- 5: Spectrum 3D Plot
- 60~67: Data Logger
- 90~105: DDP Viewer

## 2.4 MoveWindow

The MoveWindow function changes the position and dimensions of the specified window.

```
void MoveWindow(  
long nWindowID, // Window ID  
long nX, //Horizontal Position  
long nY, //Vertical Position  
long nWidth, //Width  
long nHeight, //Height  
long nRepaint //Repaint Flag  
);
```

### Parameters

*nWindowID*

Window ID. It specifies which window to operate.

- 0: Mainframe
- 1: Oscilloscope
- 2: Spectrum Analyzer
- 3: Multimeter
- 4: Generator
- 5: Spectrum 3D Plot
- 60~67: Data Logger
- 90~105: DDP Viewer

*nX*

Specifies the new position of the left side of the window.

*NY*

Specifies the new position of the top of the window

*nWidth*

Specifies the new width of the window

*nHeight*

Specifies the new height of the window

*nRepaint*

Specifies whether the window is to be repainted.

## 2.5 TileWindow

The TileWindow function tiles the windows horizontally or vertically within the mainframe of Multi-Instrument. The windows that can be tiled are: Oscilloscope, Spectrum Analyzer, Multimeter, and Spectrum 3D Plot.

```
void TileWindow(  
long nPattern, //Tile Pattern  
) ;
```

### Parameters

*nPattern*

It specifies how the windows to be tiled.

0: Horizontally

1: Vertically

## 2.6 ShowMenu

The ShowMenu function shows the specified menu.

```
void ShowMenu(  
long nMenuID //Menu ID  
)
```

Menu ID. It specifies which menu to operate.

0: Mainframe Menu.

## 2.7 HideMenu

The HideMenu function hides the specified menu.

```
void HideMenu(  
long nMenuID //Menu ID  
)
```

Menu ID. It specifies which menu to operate.

0: Mainframe Menu.

## 2.8 ShowToolbar

The ShowToolbar function shows the specified toolbar.

```
void ShowToolbar(  
long nToolbarID //Toolbar ID  
);
```

### Parameters

*nToolbarID*

Toolbar ID. It specifies which toolbar to operate

- 0: All Toolbars
- 1: Sampling Toolbar
- 2: Instrument Toolbar
- 3: Oscilloscope View Toolbar
- 4: Spectrum Analyzer View Toolbar
- 5: Multimeter View Toolbar
- 6: Spectrum 3D View Toolbar

## 2.9 HideToolbar

The HideToolbar function hides the specified toolbar.

```
void HideToolbar(  
long nToolbarID //Toolbar ID  
);
```

### Parameters

*NToolbarID*

Toolbar ID. It specifies which toolbar to operate

- 0: All Toolbars
- 1: Sampling Toolbar
- 2: Instrument Toolbar
- 3: Oscilloscope View Toolbar
- 4: Spectrum Analyzer View Toolbar
- 5: Multimeter View Toolbar
- 6: Spectrum 3D View Toolbar

## 2.10 ShowTitlebar

The ShowTitlebar function shows the specified titlebar.

```
void ShowTitlebar(  
long nTitlebarID //Titlebar ID  
);
```

## Parameters

*nTitlebarID*

Titlebar ID. It specifies which titlebar to operate

- 0: Mainframe Titlebar
- 1: Oscilloscope View Titlebar
- 2: Spectrum View Titlebar
- 3: Multimeter View Titlebar
- 4: Signal Generator Titlebar
- 5: Spectrum 3D Plot
- 60~67: Data Logger
- 90~105: DDP Viewer

## 2.11 HideTitlebar

The HideTitlebar function hides the specified toolbar.

```
void HideTitlebar(
long nTitlebarID //Titlebar ID
);
```

## Parameters

*NTitlebarID*

Titlebar ID. It specifies which titlebar to operate

- 0: Mainframe Titlebar
- 1: Oscilloscope View Titlebar
- 2: Spectrum View Titlebar
- 3: Multimeter View Titlebar
- 4: Signal Generator Titlebar

## 2.12 SetSignalGeneratorParameters

The SetSignalGeneratorParameters function sets the parameters of the signal generator.

```
void SetSignalGeneratorParameters(
long nLoopBackMode, //Loopback Mode
long nSamplingFrequency, //Sampling Frequency
long nSamplingChannels, //Sampling Channels
long nSamplingBitResolution, //Sampling Bit Resolution
long nWaveformA, //Waveform for Channel A
double dFrequencyA, //Frequency for Channel A
double dAmplitudeA, //Amplitude for Channel A
long nWaveformB, //Waveform for Channel B
double dFrequencyB, //Frequency for Channel B
double dAmplitudeB, //Amplitude for Channel B
double dPhaseDifference, //Phase Difference between Channels
//A & B
long nMaskMode, //Mask Mode
```

```

double dMaskOn, //Mask On Period
double dMaskOff, //Mask Off Period
long nFadeMode, //Fade Mode
double dFadeIn, //Fade In Period
double dFadeOut, //Fade Out Period
double dTotalDuration, //Total Duration
long nLoop, //Loop Flag
long nSweepMode, //Sweep Mode
double dStartA, //Start Frequency/Amplitude for Channel A
double dEndA, //End Frequency/Amplitude for Channel A
double dStartB, //Start Frequency/Amplitude for Channel B
double dEndB, //End Frequency/Amplitude for Channel B
long nLinearLogA, //Linear/Log Sweep Flag for Channel A
long nLinearLogB, //Linear/Log Sweep Flag for Channel B
double dDutyCycleA, //Duty Cycle for Channel A
double dDutyCycleB, //Duty Cycle for Channel B
long nMLSLength //MLS Length
);

```

### **Parameters**

#### *nLoopBackMode*

Loopback Mode:

- 0: No Loopback
- 1: iA=oA, iB=oB
- 2: iA=oA, iB=oA
- 3: iB=oA
- 4: Sync. No Loopback
- 5: Sync. iB=oA
- 6: Sync. iB←oA

#### *nSamplingFrequency*

Sampling Frequency:

- 0: 2kHz
- 1: 4kHz
- 2: 8kHz
- 3: 11.025kHz
- 4: 16kHz
- 5: 22.05kHz
- 6: 32kHz
- 7: 44.1kHz
- 8: 48kHz
- 9: 64kHz
- 10: 88.2kHz
- 11: 96kHz
- 12: 176.4kHz
- 13: 192kHz
- 14: 200kHz

Note: The above options are valid if sound card MME is chosen as the DAC device in Multi-Instrument. Otherwise, the indexed sampling frequencies may be

different from the above. Please check the Multi-Instrument software to see the options available for a specific DAC device.

#### *nSamplingChannels*

Sampling Channels:

- 0: A
- 1: A&B

Note: The above options are valid if sound card MME is chosen as the DAC device in Multi-Instrument. Otherwise, the indexed sampling channels may be different from the above. Please check the Multi-Instrument software to see the options available for a specific DAC device.

#### *nSamplingBitResolution*

Sampling Bit Resolution:

- 0: 8 Bit
- 1: 16 Bit
- 2: 24 Bit

Note: The above options are valid if sound card MME is chosen as the DAC device in Multi-Instrument. Otherwise, the indexed sampling bit resolution may be different from the above. Please check the Multi-Instrument software to see the options available for a specific DAC device.

#### *nWaveformA*

Waveform for Channel A:

- 0: None
- 1: Sine
- 2: Rectangle
- 3: Triangle
- 4: SawTooth
- 5: WhiteNoise
- 6: PinkNoise
- 7: MultiTones
- 8: Arbitrary (via Waveform Library File)
- 9: MLS

Note: if *nWaveformA* is set to 7 or 8, *nWaveformB* should also be set to 7 or 8 respectively, too.

#### *dFrequencyA*

Frequency value in Hz for Channel A. It must be less than or equal to  $\frac{1}{2}$  of the sampling frequency.

#### *dAmplitudeA*

Amplitude value in DAC engineering unit for Channel A. It must be within the full-scale DAC range.

#### *nWaveformB*

Waveform for Channel B:

- 0: None
- 1: Sine
- 2: Rectangle
- 3: Triangle
- 4: SawTooth
- 5: WhiteNoise
- 6: PinkNoise
- 7: MultiTones
- 8: Arbitrary (via Waveform Library File)
- 9: MLS

Note: if nWaveformB is set to 7 or 8, nWaveformA should also be set to 7 or 8 respectively, too.

*dFrequencyB*

Frequency value in Hz for Channel B. It must be less than or equal to  $\frac{1}{2}$  of the sampling frequency.

*dAmplitudeB*

Amplitude value in DAC engineering unit for Channel B. It must be within the full-scale DAC range.

*dPhaseDifference*

Phase Difference value in degree between the two channels. It must be in the range of  $-180\sim 180$  degree.

*nMaskMode*

Mask Mode:

- 0: No Mask
- 1: Mask with Phase Lock
- 2: Mask with No Phase Lock

*dMaskOn*

Mask On period in second.

*dMaskOff*

Mask Off period in second.

*nFadeMode*

Fade Mode:

- 0: No Fade
- 1: Fade

*dFadeIn*

Fade In period in second.

*dFadeOut*

Fade Out period in second.

*dTotalDuration*

Total Duration in second. Under non-sweep mode, it specifies the duration of the output signal, if nLoop=0. Under sweep mode, it specifies the sweep duration instead.

#### *nLoop*

It specifies whether the output signal should be repeated in loop. Under non-sweep mode, the Total Duration will be ignore if nLoop=1.

- 0: No Loop
- 1: Loop

#### *nSweepMode*

Sweep mode:

- 0: No Sweep
- 1: Frequency Sweep
- 2: Amplitude Sweep

#### *dStartA*

Start frequency in Hz under frequency sweep mode for Channel A, or start amplitude in DAC engineering unit under amplitude sweep mode for Channel A.

#### *dEndA*

End frequency in Hz under frequency sweep mode for Channel A, or end amplitude in DAC engineering unit under amplitude sweep mode for Channel A.

#### *dStartB*

Start frequency in Hz under frequency sweep mode for Channel B, or start amplitude in DAC engineering unit under amplitude sweep mode for Channel B.

#### *dEndB*

End frequency in Hz under frequency sweep mode for Channel B, or end amplitude in DAC engineering unit under amplitude sweep mode for Channel B.

#### *nLinearLogA*

Linear/Log sweep Mode for channel A:

- 0: Linear
- 1: Log

#### *nLinearLogB*

Linear/Log sweep Mode for channel B:

- 0: Linear
- 1: Log

#### *dDutyCycleA*

Duty Cycle (0~100) for Channel A, applicable only if the waveform is rectangle.

#### *dDutyCycleB*

Duty Cycle (0~100) for Channel B, applicable only if the waveform is rectangle.

#### *nMLSLength*

MLS length, applicable only if the waveform is MLS.

0: 127  
1: 225  
2: 511  
3: 1023  
4: 2047  
5: 4095  
6: 8191  
7: 16383  
8: 32767  
9: 65535  
10: 131071  
11: 262143  
12: 524287  
13: 1048575  
14: 2097151  
15: 4194303  
16: 8388607  
17: 16777215

### **Remarks**

This function can only be called after the signal generator is opened. Calling this function will have no effect when the signal generator of Multi-Instrument is in running state.

## **2.13 UpdateMultiToneItem**

The UpdateMultiToneItem function updates the specified Multitone item.

```
void UpdateMultiToneItem(  
long nChannelNo, //Channel No.  
long nItemNo, //Item No.  
long nWaveform, //Waveform  
long nFrequency, //Frequency  
double dRelativeAmplitude, //Relative Amplitude  
double dPhase //Phase  
);
```

### **Parameters**

*nChannelNo*

Channel No.:

0: Channel A

1: Channel B

*nItemNo*

Item No.. It must be in the range of 0~31.

*nWaveform*

Waveform:

1: Sine

2: Rectangle

3: Triangle

- 4: SawTooth
- 5: WhiteNoise
- 6: PinkNoise

*nFrequency*

Frequency value in Hz. It must be less than or equal to  $\frac{1}{2}$  of the sampling frequency and greater than 0 Hz.

*dRelativeAmplitude*

Relative Amplitude value. It is recommended to use a value in the range of 0~1. It is dimensionless.

*dPhase*

Initial Phase value in degree. It must be in the range of -180~180 degree.

**Remarks**

This function should be called just after calling SetSignalGeneratorParameters function to set the waveform for both channels to be MultiTones.

**2.14 SetNumberOfMultiToneItems**

The SetNumberOfMultiToneItems function sets the number of MultiTone items for both Channels A & B.

```
void SetNumberOfMultiToneItems (
long nCountA,      //Number of MultiTone items in Channel A
long nCountB      //Number of MultiTone items in Channel B
);
```

**Parameters***nCountA*

Number of MultiTone items in Channel A. It must be in the range of 0~31;

*nCountB*

Number of MultiTone items in Channel B. It must be in the range of 0~31;

**Remarks**

This function should be called just after calling SetSignalGeneratorParameters function to set the waveform for both channels to be MultiTones.

**2.15 LoadWFLibrary**

The LoadWFLibrary function loads the waveform library file.

```
void LoadWFLibrary (
LPCTSTR sWFLibraryFileName    //Waveform library file name
);
```

### **Parameters**

*SWFLibraryFileName*

Waveform library file name.

### **Remarks**

This function should be called just after calling SetSignalGeneratorParameters function to set the waveform for both channels to be WFLibrary.

## **2.16 StartDAO**

The StartDAO function starts the signal generator.

```
void StartDAO()
```

## **2.17 StopDAO**

The StopDAO function stops the signal generator.

```
void StopDAO()
```

## **2.18 GetDAOStatus**

The GetDAOStatus function returns the DAO status. DAO is the short form for “Data Output”.

```
long GetDAOStatus()
```

### **Return Values**

0: Stopped

1: Running

## **2.19 SetDAQParameters**

The SetDAQParameters function sets the DAQ parameters. DAQ is the short form for “Data Acquisition”.

```
SetDAQParameters(  
long nSamplingFrequency, //Sampling Frequency  
long nSamplingChannels, //Sampling Channels  
long nSamplingBitResolution, //Sampling Bit Resolution  
long nRecordLength, //Record Length  
long nTriggerMode, //Trigger Mode  
long nTriggerSource, //Trigger Source  
long nTriggerEdge, //Trigger Edge
```

```

long nTriggerLevel, //Trigger Level
long nTriggerDelay, //Trigger Delay
long nRangeA, //Range for Channel A
long nRangeB, //Range for Channel B
long nCouplingTypeA, //Coupling Type for Channel A
long nCouplingTypeB, //Coupling Type for Channel B
long nProbeSwitchPositionA, //Probe Switch Position for Ch. A
long nProbeSwitchPositionB //Probe Switch Position for Ch. B
)

```

### **Parameters**

*nSamplingFrequency*

Sampling Frequency:

- 0: 2kHz
- 1: 4kHz
- 2: 8kHz
- 3: 11.025kHz
- 4: 16kHz
- 5: 22.05kHz
- 6: 32kHz
- 7: 44.1kHz
- 8: 48kHz
- 9: 64kHz
- 10: 88.2kHz
- 11: 96kHz
- 12: 176.4kHz
- 13: 192kHz
- 14: 200kHz

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling frequencies may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

*nSamplingChannels*

Sampling Channels:

- 0: A
- 1: A&B

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling channels may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

*nSamplingBitResolution*

Sampling Bit Resolution:

- 0: 8 Bit
- 1: 16 Bit
- 2: 24 Bit

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling bit resolution may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

#### *nRecordLength*

Record Length per frame of the oscilloscope.

#### *nTriggerMode*

Trigger Mode:

- 0: Auto
- 1: Normal
- 2: Single
- 3: Slow

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling bit resolution may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

#### *nTriggerSource*

Trigger Source:

- 0: A
- 1: B

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling bit resolution may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

#### *nTriggerEdge*

Trigger Source:

- 0: Up
- 1: Down
- 2: Up or Down
- 3: Jump
- 4: Differential

Note: The above options are valid if sound card MME is chosen as the ADC device in Multi-Instrument. Otherwise, the indexed sampling bit resolution may be different from the above. Please check the Multi-Instrument software to see the options available for a specific ADC device.

#### *nTriggerLevel*

Trigger Level, adjustable from -100%~100% depending on the ADC device used.

#### *nTriggerDelay*

Trigger Delay, adjustable from -100%~100% depending on the ADC device used.

#### *nRangeA*

ADC Range for Channel A, available options depending on the ADC device used.

*nRangeB*

ADC Range for Channel B, available options depending on the ADC device used:

*nCouplingTypeA*

Coupling Type for Channel A, available options depending on the ADC device used:

0: AC

1: DC

*nCouplingTypeB*

Coupling Type for Channel B, available options depending on the ADC device used:

0: AC

1: DC

*nProbeSwitchPositionA*

Probe Switch Position for Channel A, depending on the probe used.

0: 1

1: 2

2: 3

*nProbeSwitchPositionB*

Probe Switch Position for Channel B, depending on the probe used.

0: 1

1: 2

2: 3

### **Remarks**

Calling this function will have no effect when the DAQ of Multi-Instrument is in running state.

## **2.20 SetViewParameters**

The SetViewParameters sets the parameters of different views of Multi-Instrument.

```
SetViewParameters(  
long nViewID, //View ID  
long nViewType, //View Type  
long nParameter1, //Parameter 1  
long nParameter2, //Parameter 2  
long nParameter3, //Parameter 3  
long nParameter4, //Parameter 4  
long nParameter5, //Parameter 5  
long nParameter6, //Parameter 6  
long nParameter7, //Parameter 7  
long nParameter8, //Parameter 8  
long nParameter9, //Parameter 9  
long nParameter10, //Parameter 10
```

```
double dParameter11, //Parameter 11
double dParameter12 //Parameter 12
)
```

### **Parameters**

*nViewID*

View ID:

- 0: Oscilloscope
- 1: Spectrum Analyzer
- 2: Multimeter
- 3: Spectrum 3D Plot

*nViewType*

View Type:

For Oscilloscope:

- 0: A&B
- 1: A+B
- 2: A-B
- 3: A×B
- 4: A|B

For Spectrum Analyzer:

- 0: Amplitude Spectrum
- 1: Phase Spectrum
- 2: Auto Correlation
- 3: Cross Correlation
- 4: Coherence Function
- 5: Transfer Function
- 6: Impulse Response

For Multimeter

- 0: RMS
- 1: dBV
- 2: dBu
- 3: dBSPL
- 4: dB(A)
- 5: dB(B)
- 6: dB(C)
- 7: Frequency Counter
- 8: RPM
- 9: Counter
- 10: Duty Cycle
- 11: F/V
- 12: Cycle RMS
- 13: Cycle Mean
- 14: Vibrometer

For Spectrum 3D Plot

- 0: Waterfall

## 1: Spectrogram

### *nParameter1*

#### View Parameter 1:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Index of FFT Size options.

For Multimeter: Counter Trigger Level (-100%~100%) for Channel A.

For Spectrum 3D Plot: Index of T Range options.

### *nParameter2*

#### View Parameter 2:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Index of Window Function options.

For Multimeter: Counter Trigger Level (-100%~100%) for Channel B.

For Spectrum 3D Plot: Tilt Angle (0~90 degree) of T axis

### *nParameter3*

#### View Parameter 3:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Index of Window Overlap options.

For Multimeter: Counter Trigger Hysteresis (0%~100%) for Channel A.

For Spectrum 3D Plot: Height Percentage (5%~90%) of Y axis

### *nParameter4*

#### View Parameter 4:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Reserved.

For Multimeter: Counter Trigger Hysteresis (0%~100%) for Channel B.

For Spectrum 3D Plot: Reserved.

### *nParameter5~nParameter10*

Reserved.

### *dparameter11*

#### View Parameter 11:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Reserved.

For Multimeter: Counter Frequency Divider for Channel A.

For Spectrum 3D Plot: Reserved.

### *dparameter12*

#### View Parameter 12:

For Oscilloscope: Reserved.

For Spectrum Analyzer: Reserved.

For Multimeter: Counter Frequency Divider for Channel B.

For Spectrum 3D Plot: Reserved.

## 2.21 StartDAQ

The StartDAQ function starts the DAQ.

```
void StartDAQ()
```

## 2.22 StopDAQ

The StopDAQ function stops the DAQ.

```
void StopDAQ()
```

## 2.23 Record

The Record function starts the DAQ in record mode.

```
void Record()
```

## 2.24 GetDAQStatus

The GetDAQStatus function returns the DAQ status. DAQ is the short form for “Data Acquisition”.

```
long GetDAQStatus()
```

### Return Values

0: Stopped

1: Running

## 2.25 GetDDP

The GetDDP function returns the requested DDP value. DDP is the short form for “Derived Data Point”.

```
double GetDDP(  
LPCTSTR sDDPName //DDP Name  
)
```

### Parameters

*sDDPName*

DDP name, DDP is the short form for “Derived Data Point”.

### Return Values

1.0E40: failed.

Others: the current value of the DDP.

## 2.26 LoadPanelSettingFile

The LoadPanelSettingFile function loads the specified panel setting file.

```
LoadPanelSettingFile(  
LPCTSTR sPanelSettingFileName //Panel Setting File name  
)
```

### Parameters

*sPanelSettingFileName*  
Panel Setting File name.

## 2.27 LoadFile

The LoadFile function loads the specified wave file or TXT file.

```
LoadFile(  
LPCTSTR sFileName //File name  
)
```

### Parameters

*sFileName*  
File name to be loaded.

## 2.28 SaveFile

The SaveFile function saves the wave data to the specified file.

```
SaveFile(  
LPCTSTR sFileName //File name  
)
```

### Parameters

*sFileName*  
File name to be saved.

## 2.29 OscilloscopeExport

The OscilloscopeExport function exports the wave data to the specified TXT file.

```
OscilloscopeExport(  
LPCTSTR sFileName //File name  
)
```

### Parameters

*sFileName*

File name to be exported.

## 2.30 SpectrumAnalyzerExport

The SpectrumAnalyzerExport function exports the data in the spectrum analyzer to the specified TXT file.

```
SpectrumAnalyzerExport(  
LPCTSTR sFileName //File name  
)
```

### Parameters

*sFileName*

File name to be exported.

## 2.31 LockGUI

The LockGUI function locks the GUI of Multi-Instrument. GUI is the short form for “Graphical User Interface”.

```
void LockGUI()
```

## 2.32 UnlockGUI

The UnLockGUI function unlocks the GUI of Multi-Instrument. GUI is the short form for “Graphical User Interface”.

```
void UnlockGUI()
```

## 2.33 Unlock

The Unlock function unlocks the MI automation server so that it can be controlled by an Automation client program via the its exposed interfaces. This function must be called before any Automation interfaces can be used.

```
void Unlock(  
long nSerialNumberPart1, //serial number part 1  
long nSerialNumberPart2, //serial number part 2  
long nSerialNumberPart3, //serial number part 3  
long nSerialNumberPart4 //serial number part 4  
)
```

### Parameters

*nSerialNumberPart1*

Part 1 of the serial number of the MI automation server.

*nSerialNumberPart1*

Part 2 of the serial number of the MI automation server.

*nSerialNumberPart1*

Part 3 of the serial number of the MI automation server.

*nSerialNumberPart1*

Part 4 of the serial number of the MI automation server.

Note that:

1. The serial number has a format of part1-part2-part3-part4, where each part contains four characters in hex format
2. For copy-protected MI automation server, such as the trial version, the softkey activated version, or the USB hardkey activated version, a generic serial number F65A-7C8A-D92E-18EC should be used.
3. For not-copy-protected MI automation server, which is usually the case for OEM, a customer specific serial number will be given when the server is purchased from Virtins Technology.

## **3. Multi-Instrument Automation Client Program Development Guide**

### **3.1 Basic Files of Multi-Instrument Automation Server**

For OEM customer, the following files are the basic files required to run the Multi-Instrument Automation Server:

- (1) MI.exe
- (2) Scins.cfg
- (3) VScopeResENUS.dll
- (4) ADCDevice.ddb
- (5) DACDevice.ddb
- (6) SoundCardMMEDAQ.dll
- (7) SoundCardMMEDAO.dll

The type library file:

- (1) MI.tlb

is also provided, which can be used during client program development.

### 3.2 Multi-Instrument Automation Server Registration

An Automation Server must be registered in the system first before it can be used by a client program.

To register Multi-Instrument Automation Server, run the following command:

```
MI /regserver
```

The automation server will be automatically registered if you run the Multi-Instrument program once in the system.

To un-register Multi-Instrument Automation Server, run the following command:

```
MI /unregserver
```

Under Windows Vista, these commands must be run with the admin right. To run the command line with admin right, simply click the Start button and type cmd in the Instant Search field, and press CTRL+SHIFT+ENTER instead of just ENTER.

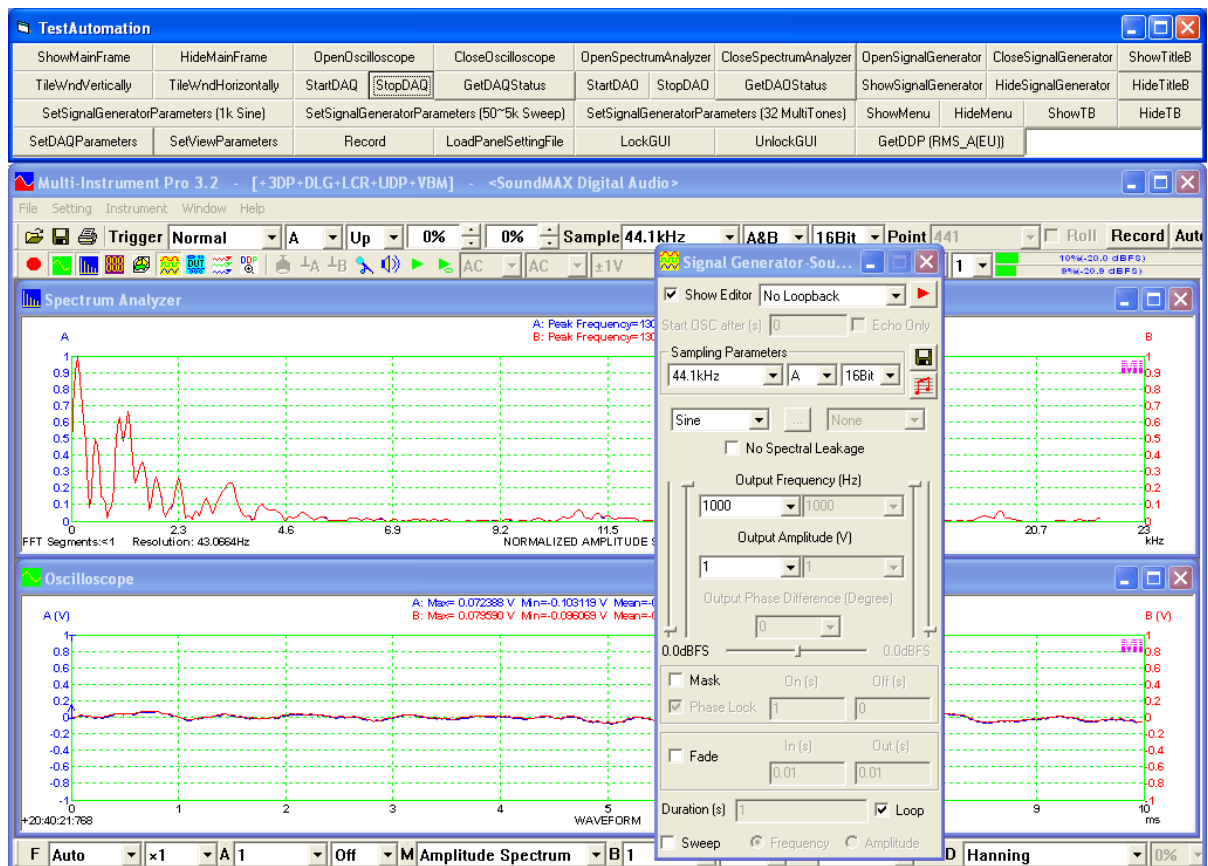
### 3.3 Multi-Instrument Automation Server CLSID and ProgID

CLSID: 688C2F6F-CA63-4D40-83D9-C30CA6524780

ProgID: MI.Automation

## 4. Sample Automation Client Programs

### 4.1 TestAutomation written in Visual Basic 6.0



The program TestAutomation demonstrates how to use the Automation interfaces exposed by the Multi-Instrument Automation Server to control the Multi-Instrument program. There are four lines of push buttons on the top of Multi-Instrument program (see screenshot above). The labels on these buttons are self-explanatory. You can use these buttons to control the Multi-Instrument program.

To launch the Automation Server in Visual Basic, only two lines of codes are required:

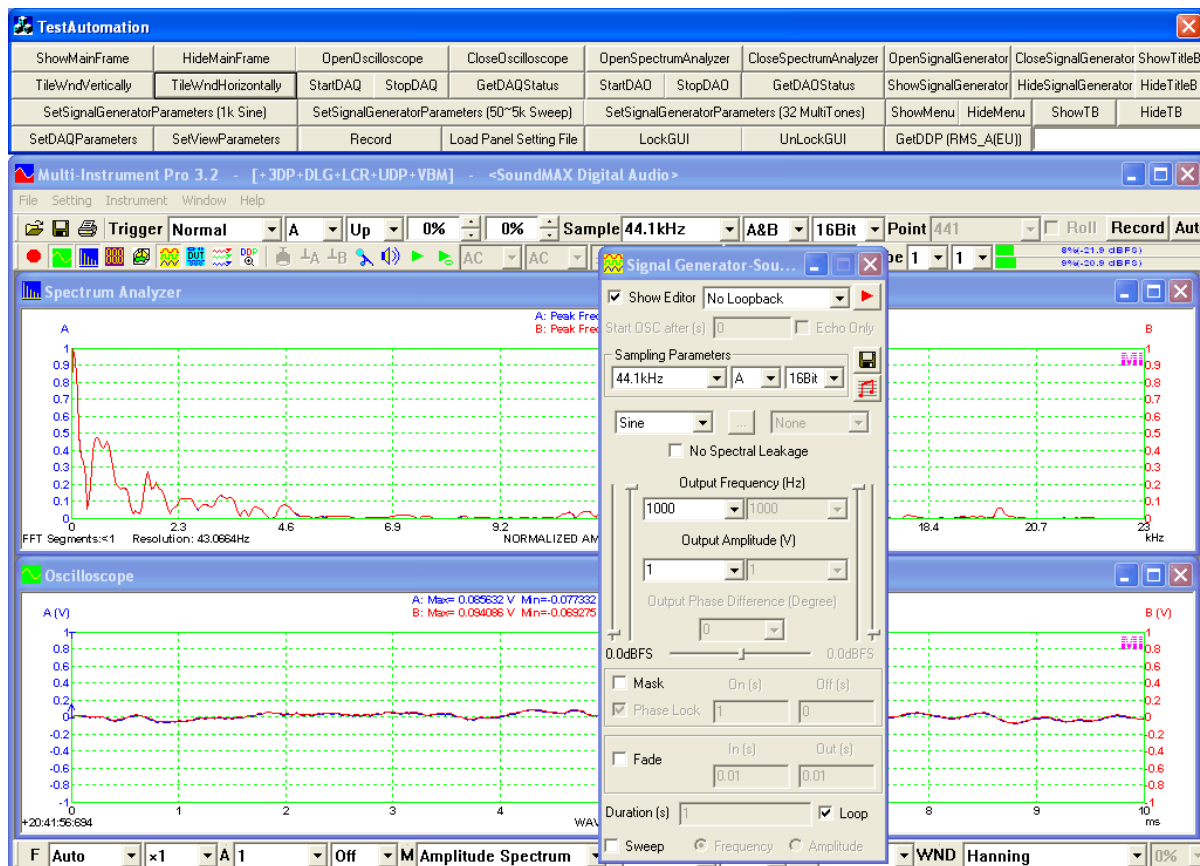
```
Dim TestMI As Object
Set TestMI = CreateObject("MI.Automation")
```

After that, you can use any Automation interfaces by simply calling:

```
Call TestMI.xxxx
```

where xxxx is the interfacing API documented in Chapter 2. Note that you must call the Unlock function with the correct serial number first before you can use other interfacing APIs.

## 4.2 TestAutomation written in Visual C++ 6.0



It has the same functionality as its counterpart in Visual Basic, as shown above. Among the project files, there are two files named: MI.h and MI.cpp. They were generated by “Add Class” from the type library “MI.tlb” using the MFC ClassWizard.

To launch the Automation Server in Visual C++, only three lines of codes are required

- (1) `AfxOleInit()` //It is in `InitInstance()` in `TestAutomation.cpp`  
It is used to initialize the OLE library.
- (2) `IAutomation m_Automation` //It is in `TestAutomationDlg.h`
- (3) `m_Automation.CreateDispatch("MI.Automation")`  
// It is in `OnInitDialog()` in `TestAutomationDlg.cpp`

The above (2) and (3) creates an `IDispatch` object and attaches it to the `COleDispatchDriver` object. After this, you can use any Automation interfaces by simply calling:

```
M_Automation.xxxx
```

where `xxxx` is the interfacing API documented in Chapter 2. Note that you must call the `Unlock` function with the correct serial number first before you can use other interfacing APIs.