

vtDAQ and vtDAO APIs

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

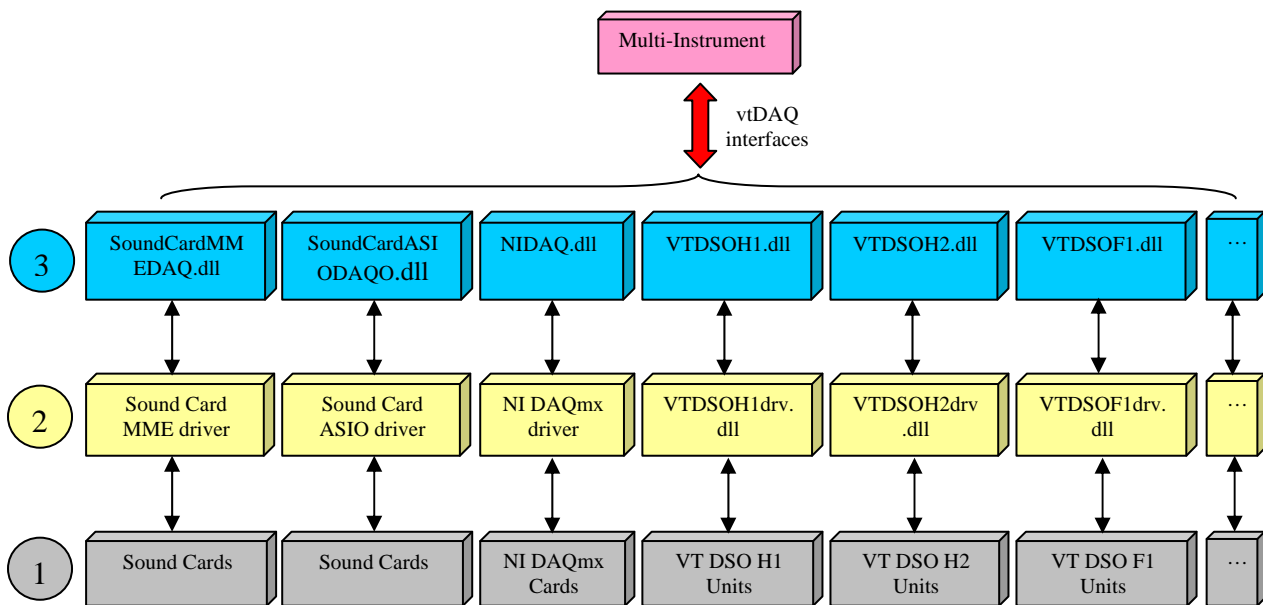
1. INTRODUCTION	4
2. VTDAQ INTERFACE SPECIFICATIONS	6
2.1 STRUCTURE DEFINITION	6
2.1.1 <i>SamplingParametersStruct</i>	6
2.1.2 <i>TriggerParametersStruct</i>	8
2.1.3 <i>DAQDataStruct</i>	10
2.1.4 <i>DAQDAOSyncParametersStruct</i>	12
2.1.5 <i>DAQDeviceInfoStruct</i>	12
2.2 APIs	16
2.2.1 <i>DAQ_SetParameters</i>	16
2.2.2 <i>DAQ_Start</i>	17
2.2.3 <i>DAQ_Stop</i>	17
2.2.4 <i>DAQ_GetSamplePosition</i>	17
2.2.5 <i>DAQ_GetDeviceList</i>	18
2.2.6 <i>DAQ_GetDeviceInfo</i>	18
2.2.7 <i>DAQ_Unlock</i>	19
2.2.8 <i>DAQ_Load</i>	20
2.2.9 <i>DAQ_Unload</i>	20
2.3 MESSAGES AND STATUS FLAGS.....	20
2.3.1 <i>WM_MYMESSAGE_DAO_START</i>	20
2.3.2 <i>WM_MYMESSAGE_DAO_DATA</i>	20
2.3.3 <i>WM_MYMESSAGE_DAO_STOP</i>	20
2.3.4 <i>WM_MYMESSAGE_DAO_ERROR</i>	20
3. VTDAQ DEVELOPMENT GUIDE	21
3.1 FLOWCHARTS.....	21
3.2 BASIC FILES	22
4. SAMPLE PROGRAMS	23
4.1 TESTDAQ WRITTEN IN VISUAL C++ 6.0.....	23
4.2 MYDAQ.DLL WRITTEN IN VISUAL C++ 6.0.....	23
5. VTDAO INTERFACE SPECIFICATIONS	25
5.1 STRUCTURE DEFINITION	25
5.1.1 <i>OutputSamplingParametersStruct</i>	25
5.1.2 <i>DAODataStruct</i>	27
5.1.3 <i>DAODeviceInfoStruct</i>	28
5.2 APIs	30
5.2.1 <i>DAO_SetParameters</i>	30
5.2.2 <i>DAO_Start</i>	30
5.2.3 <i>DAO_Stop</i>	31
5.2.4 <i>DAO_GetSamplePosition</i>	31
5.2.5 <i>DAO_GetDeviceList</i>	31
5.2.6 <i>DAO_GetDeviceInfo</i>	32
5.2.7 <i>DAO_Unlock</i>	32
5.2.8 <i>DAO_Load</i>	33
5.2.9 <i>DAO_Unload</i>	33
5.2.10 <i>DAO_Write</i>	33
5.3 MESSAGES AND STATUS FLAGS.....	33
5.3.1 <i>WM_MYMESSAGE_DAO_START</i>	33
5.3.2 <i>WM_MYMESSAGE_DAO_DATA</i>	33
5.3.3 <i>WM_MYMESSAGE_DAO_STOP</i>	34
5.3.4 <i>WM_MYMESSAGE_DAO_ERROR</i>	34
5.3.5 <i>WM_MYMESSAGE_DAO_STOP_REQUEST</i>	34
6. VTDAO DEVELOPMENT GUIDE	34

6.1 FLOWCHARTS.....	34
6.2 BASIC FILES.....	35
6.3 HOW TO CHOOSE CORRECT OUTPUT MODE	35
6.3.1 <i>Hardware Sampling Clock</i>	35
6.3.2 <i>Software Timed Sampling Clock</i>	35
7. SAMPLE PROGRAMS	36
7.1 TESTDAO WRITTEN IN VISUAL C++ 6.0.....	36

1. Introduction

Multi-Instrument is able to interface to many ADC and DAC devices including sound cards based on the standard data acquisition software interface specification developed by Virtins Technology: vtDAQ[®] for ADC and vtDAO[®] for DAC. DAQ is a short form for Data Acquisition and DAO is a short form for Data Output. For each category of hardware devices, an intermediate interface DLL (dynamic link library) needs to be developed according to this standard interface specification to bridge Multi-Instrument and the device's original driver or software interfaces. The software can work with any device as long as the corresponding intermediate interface DLL is provided. One interface DLL should contain either the ADC functions or DAC functions, but not both if possible, even if all of these functions are supported by one single device. This is to ensure that the ADC and DAC devices can be selected independently in Multi-Instrument. For example, you can run a DSO (Digital Storage Oscilloscope) hardware for ADC and the sound card for DAC simultaneously in Multi-Instrument.

The following diagram shows how Multi-Instrument communicates with different types of ADC hardware via the standard vtDAQ interfaces.



Legend



Hardware Specific Interfaces



vtDAQ Interfaces



ADC Devices

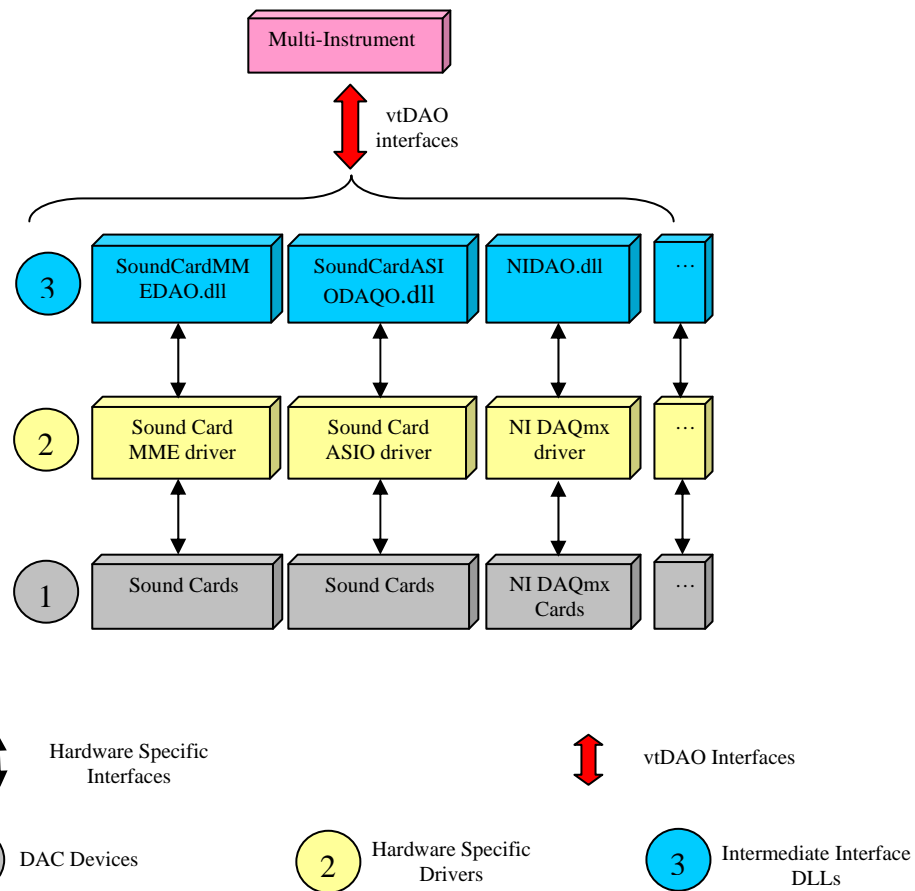


Hardware Specific Drivers



Intermediate Interface DLLs

The following diagram shows how Multi-Instrument communicates with different types of DAC hardware via the standard vtDAO interfaces.



You can write your own back-end program to interface to the ADC and DAC devices that are supported by Multi-Instrument via vtDAQ and vtDAO interfaces. **As all these devices conform to the same interface standard, you only need to write the interface codes in your program once and your program will support all these devices.**

On the other hand, you can also write your own intermediate interface DLLs to allow Multi-Instrument to interface to your own hardware. You need to perform the following steps:

- 1) Develop your own vtDAQ compatible DLL and vtDAO compatible DLL.
- 2) Delete the space-holder files in the software's root directory: MyDAQ.dll and MyDAO.dll
- 3) Name your vtDAQ compatible DLL as MyDAQ.dll and your vtDAO compatible DLL as MyDAO.dll, and put them in the software's root directory.
- 4) Remove the space-holder item "My DAQ Device" in the ADC Device Database via [Setting]>[ADC Device Database].
- 5) Remove the space-holder item "My DAO Device" in the DAC Device Database via [Setting]>[DAC Device Database].
- 6) Add your own ADC device into the above ADC Device Database
- 7) Add your own DAC device into the above DAC Device Database
- 8) Select your own ADC device via [Setting]>[ADC Device]
- 9) Select your own DAC device via [Setting]>[DAC Device]

Note that the original space-holder MyDAQ.dll and MyDAO.dll are actually the same as SoundCardMMEDAQ.dll and SoundCardMMEDAO.dll. The original space-holder “My DAQ device” and “My DAO device” are actually the sound card in your computer. These space holders are used for demonstration and testing purpose. You can replace them with your own items.

In the following chapters, the vtDAQ and vtDAO Interface Specifications will be described, followed by the following sample programs:

- TestDAQ, a sample DAQ back-end program in Visual C++ 6.0.
- TestDAO, a sample DAO back-end program in Visual C++ 6.0.
- MyDAQ, a sample DAQ intermediate interface DLL in Visual C++6.0.

2. vtDAQ Interface Specifications

2.1 Structure Definition

2.1.1 SamplingParametersStruct

```
struct SamplingParametersStruct
{
    double SamplingFrequency;
    WORD SamplingChannels;
    WORD SamplingBitResolution;
    DWORD RecordLength;
    WORD DeviceNo;
    WORD ChannelNo[32];
    double HighLimit[32];
    double LowLimit[32];
    WORD TerminalConfiguration[32];
    WORD CouplingType[32];
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```

Members

SamplingFrequency
Sampling Frequency in Hz.

SamplingChannels
Number of Sampling Channels.

SamplingBitResolution
Bit resolution of the DAQ data. It can only be 8, 16, 24, or 32 bits. It should generally be equal to the bit resolution of the ADC device. However, if the bit resolution of the ADC device is not an integer multiple of 8, then it is the intermediate DLL’s responsibility to round up the bit resolution of the raw data to the nearest integer multiple of 8. For example, NI USB-

6009 has a bit resolution of 14 when connected in differential input mode, this parameter should then be set to 16, and the NIDAQ.dll will output the DAQ data with a bit resolution of 16.

RecordLength

Record Length. It should not exceed the buffer size of the device. For devices that support continuous data streaming, the buffer size can be considered as unlimited.

DeviceNo

Device No. of the same category of ADC devices present in the system. If there is only one such device in the system, then its value should be zero. If there are multiple such devices in the system, then this parameter specifies which one to use. For example, for SoundCardMMEDAQ.dll, it specifies which sound card to use under Windows OS before Windows Vista; It specifies which endpoint (i.e. which input source (e.g. Mic, Line In, etc.) of which sound card) to use under Windows Vista and 7.

ChannelNo[32]

This array assigns each sampling channel with a physical channel No.. The sampling channel numbers must start from 0 to *SamplingChannels-1*, and each sampling channel must be assigned with a physical channel No.. A physical channel is a channel in the ADC device. For example, if the ADC device supports 16 channels, and you want to sample only Channel 5 and Channel 9 out of the 16 channels, then you should specify:

ChannelNo[0] = 5

ChannelNo[1] = 9

These parameters are not used by SoundCardMMEDAQ.dll.

For SoundCardASIODAQO.dll, ChannelNo[16] is the physical channel No. of the DAO output channel A, and ChannelNo[17] is the physical channel No. of the DAO output channel B.

HighLimit[32]

This array specifies the ADC high limit of each sampling channel.

These parameters are not used by SoundCardMMEDAQ.dll and SoundCardASIODAQO.dll.

LowLimit[32]

This array specifies the ADC low limit of each sampling channel.

These parameters are not used by SoundCardMMEDAQ.dll and SoundCardASIODAQO.dll.

TerminalConfiguration[32]

This array specifies the terminal configuration of each sampling channel.

For example, for NI DAQmx devices:

- 0: At run time, NI-DAQmx chooses the default terminal configuration for the channel.
- 1: Referenced single-ended mode
- 2: Nonreferenced single-ended mode
- 3: Differential mode
- 4: Pseudodifferential mode

These parameters are not used by SoundCardMMEDAQ.dll, SoundCardASIODAQO.dll, VTDSOH1.dll, VTDSOH2.dll, and VTDSOF1.dll.

CouplingType[32]

This array specifies the coupling type of each sampling channel.

- 0: AC
- 1: DC
- 2: GND

These parameters are not used by SoundCardMMEDAQ.dll, SoundCardASIODAQO.dll.

ReservedDouble[8]

Reserved.

ReservedDWORD[8]

Reserved.

For VT DSO-2810F, ReservedDWORD[0]=1;

For SoundCardASIODAQO.dll, ReservedDWORD[0] is the index for ASIO buffer size selection, with 0: Auto; 1: Max; 2: Min.

2.1.2 TriggerParametersStruct

```
struct TriggerParametersStruct
{
    WORD   TriggerMode;
    WORD   TriggerSource;
    WORD   TriggerEdge;
    double TriggerLevelPercent;
    double TriggerDelayPercent;
    WORD   ExtChannelNo;
    BOOL   RecordMode;
    BOOL   HardwareTrigger;
    double ReservedDouble[8];
    DWORD  ReservedDWORD[8];
};
```

Members

TriggerMode

Trigger Mode.
0: Auto or Free Run
1: Normal

TriggerSource
Trigger Source, to be assigned with the sampling channel No..
0: Channel A
1: Channel B
2: EXT
3: ALT

TriggerEdge
Trigger Edge.
0: Up
1: Down
2: Up or Down
3: Jump
4: Differential

Please refer to multi-instrument software manual for detailed explanation on the meaning of each item.

TriggerLevelPercent
Trigger Level Percentage.

For ALT mode, it is Trigger Level Percentage for Channel A.

TriggerDelayPercent
Trigger Delay Percentage.

ExtChannelNo
Physical Channel No. for External Trigger.

RecordMode
Record Mode.
FALSE: Not Record Mode
The specified trigger parameters will have effect on acquiring each frame of data.
TRUE: Record Mode.
The specified trigger parameters will only have effect on acquiring the first frame of data, and subsequent frames of data will be acquired continuously regardless of the trigger parameters.

HardwareTriggers
FALSE: Not hardware trigger. Software trigger is only possible for those ADC devices that support continuous data streaming.
TRUE: Hardware trigger.

ReservedDouble[8]
Reserved.

For ALT mode, ReservedDouble[0] is Trigger Level Percent for Channel B.

```
ReservedDWORD[8]
    Reserved.
```

2.1.3 DAQDataStruct

```
struct DAQDataStruct
{
    char ** ppRecordData;
    long * pRecordDataCount;
    unsigned long RecordBufferCount;
    DWORD RecordedTriggerLevelBefore;
    DWORD RecordedTriggerLevelAfter;
    DWORD RecordedTriggerLevel;
    struct _timeb RecordedTimeStamp;
    WORD Status;
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```

Members

ppRecordData

Pointer to a pointer array. Each element of the pointer array contains the address of a “char” data array. The “char” data array is called a record buffer. The size of each record buffer must be equal to [Record Length] × [Sampling Channels] × [Sampling Bit Resolution] / 8. Note that for 8-bit data, the data are stored in the record buffer as unsigned values; for 16-bit, 24-bit, and 32-bit data, the data are stored in the record buffer as signed values. This conforms to the data format in a wave file. Therefore it can be readily stored in a wave file without converting the data format.

pRecordDataCount

Pointer to a “long” data array, where each element of the array contains the actual number of recorded bytes in the corresponding record buffer. The size of this array must be equal to the number of record buffers. **After the calling program processes all the recorded data into the record buffer, it must set the number of recorded bytes to zero to inform the intermediate DLL that this record buffer is ready to accept new data.**

RecordBufferCount

Number of record buffers. In most of cases, one record buffer is enough. In some special cases, especially if the calling program is not able to process the recorded data in time, you may need to create and use more record buffers to buffer the recorded data before they can be processed.

RecordedTriggerLevelBefore

The value of the sample just before the first recorded sample. This parameter is only valid if software trigger is used. The value is stored as an unsigned value.

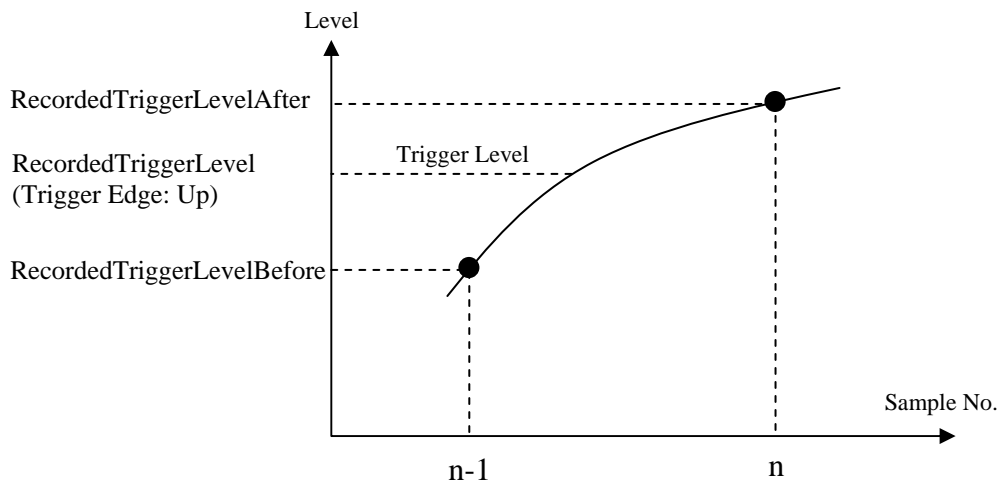
RecordedTriggerLevelAfter

The value of the first recorded sample. This parameter is only valid if software trigger is used. The value is stored as an unsigned value.

RecordedTriggerLevel

Trigger level. This parameter is only valid if software trigger is used. The value is stored as an unsigned value.

The relationship of *RecordedTriggerLevelBefore*, *RecordedTriggerLevelAfter*, *RecordedTriggerLevel* is shown in the following figure.



RecordedTimeStamp

Time stamp of the first sample in the record buffer.

Status

DAQ status

Bit0: 0: Not Triggered; 1: Triggered

Bit1: 0: Stop; 1: Running

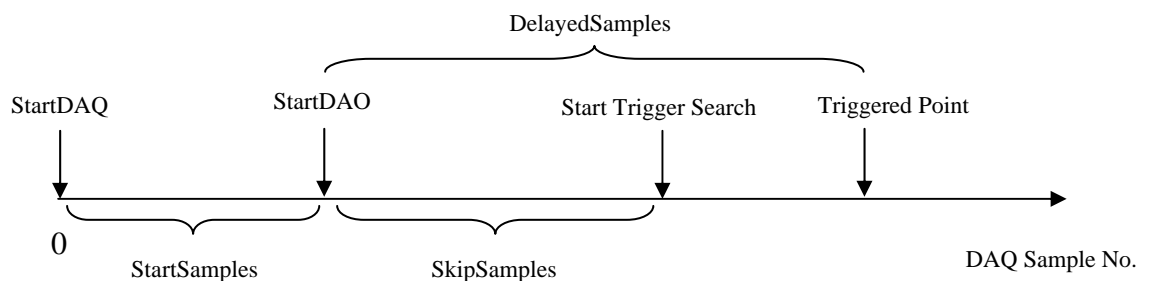
ReservedDouble[8]

Reserved.

ReservedDWORD[8]
Reserved.

2.1.4 DAQDAOSyncParametersStruct

```
struct DAQDAOSyncParametersStruct
{
    long StartSamples;
    long SkipSamples;
    int Mode;
    long DelayedSamples;
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```



The above figure shows the timing diagram of the DAQ and DAO synchronization process. The DAQ should be started first by the calling program with *StartSamples*=-1. When the calling program starts the DAO, it should get the DAQ sample position at the same time using `DAQ_GetSamplePosition()` and assign its value to *StartSamples*. The calling program can also specify how many samples to be skipped (via *SkipSamples*) before the interface DLL starts to search for the trigger event. The interface DLL will calculate the *DelayedSamples* after the trigger event is found.

Mode

4: Sync. No Loopback

5: Sync. iB = oA

6: Sync. iB \leftarrow oA

Please refer to the Signal Generator chapter in the Multi-Instrument software manual for the definition of these modes.

2.1.5 DAQDeviceInfoStruct

```
struct DAQDeviceInfoStruct
{
    char ProductCategory[255];
    char ProductType[255];
};
```

```

char ProductNumber[255];
char DeviceSerialNumber[255];
char ChassisModuleName[255];
BOOL SoftwareTriggerSupported;
BOOL HardwareTriggerSupported;
BOOL HardwareTriggerLevelAdjustable;
BOOL HardwarePreTriggerSupported;
BOOL HardwareALTTriggerSupported;
BOOL ExternalTriggerSupported;
BOOL ExternalTriggerLevelAdjustable;
WORD BasicUnit;
WORD NumberOfAIs;
double SingleChannelRate;
double MultiChannelRate;
double MinimumRate;
BOOL SimultaneousSamplingSupported;
double VoltageRange[64];
double FrequencyRange[64];
WORD BitRange[32];
long CouplingType[3];
BOOL CouplingTypePerChannel;
long TerminalType[5];
DWORD BufferSize;
BOOL Validity;
double ReservedDouble[8];
DWORD ReservedDWORD[8];
};

```

Parameters

ProductCategory[255]

Product Category name.

ProductType[255]

Product Type name.

ProductNumber[255]

Product Number.

DeviceSerialNumber[255]

Device Serial Number.

ChassisModuleName[255]

Chassis Module Name.

SoftwareTriggerSupported

Indicates if the device supports software trigger. Only those devices that support continuous data streaming can support software trigger.

FALSE: Not supported.

YES: Supported.

HardwareTriggerSupported

Indicates if the device supports hardware trigger.

FALSE: Not supported.

YES: Supported.

HardwareTriggerLevelAdjustable

Indicates if the hardware trigger level is adjustable.

FALSE: Not supported.

YES: Supported.

HardwarePreTriggerSupported

Indicates if the hardware pre-trigger is supported.

FALSE: Not supported.

YES: Supported.

HardwareALTTriggerSupported

Indicates if the device supports hardware ALT trigger.

FALSE: Not supported.

YES: Supported.

ExternalTriggerSupported

Indicates if the device supports hardware EXT trigger.

FALSE: Not supported.

YES: Supported.

ExternalTriggerLevelAdjustable

Indicates if the external trigger level is adjustable.

FALSE: Not adjustable.

YES: Adjustable.

BasicUnit

Indicates the ADC type of the device.

0: Analog voltage to digital conversion.

1: Analog current to digital conversion.

NumberOfAIs

Indicates the number of AI input channels of the device.

SingleChannelRate

Indicates the maximum sampling rate of a channel if only a single input channel of the device is used.

MultiChannelRate

Indicates the maximum sampling rate of a channel if all the input channels of the device are used. For multiplexed devices, $\text{MultiChannelRate} = \text{SingleChannelRate} / \text{NumberOfAIs}$

MinimumRate

Indicates the minimum sampling rate of a channel of the device.

SimultaneousSamplingSupported

Indicates if the device supports simultaneous sampling.

VoltageRange[64]

Indicates the pairs of input voltage ranges supported by the device. Each pair consists of a low limit, followed by a high limit. The pairs are stored in ascending order. The not-used elements of this array will be filled with zeros.

FrequencyRange[64]

Indicates the pairs of input sampling frequency ranges supported by this device. Each pair consists of a low limit, followed by a high limit. The pairs are stored in ascending order. The not-used elements of this array will be filled with zeros.

BitRange[32]

Indicates input bit resolutions supported by the device. The values are stored in ascending order. The not-used elements of this array will be filled with zeros.

CouplingType[3]

Indicates the coupling types supported by the device.

- 0: AC
- 1: DC
- 2: GND

The not-used elements of this array will be filled with -1.

CouplingTypePerChannel

Indicates if the device supports change of coupling type on per channel basis.

TerminalType[5]

Indicates the coupling types supported by the device.

- 0: default.
- 1: Referenced single-ended mode
- 2: Nonreferenced single-ended mode
- 3: Differential mode
- 4: Pseudodifferential mode

The not-used elements of this array will be filled with -1.

BufferSize

Indicates the buffer size (in samples) per channel. A value of 4294967295 indicates that there is no limit on the buffer size.

Validity

Reserved.

ReservedDouble[8]

Reserved.

ReservedDWORD[8]

Reserved.

2.2 APIs

2.2.1 DAQ_SetParameters

The DAQ_SetParameters function sets the DAQ parameters.

```
int DAQ_SetParameters(
    SamplingParametersStruct& SamplingParameters,
    TriggerParametersStruct& TriggerParameters,
    DAQDataStruct& DAQData,
    DAQDAOSyncParametersStruct& DAQDAOSyncParameters,
    DWORD dwCallback,
    DWORD fdwOpen
);
```

Parameters

SamplingParameters

Address of a SamplingParametersStruct structure that contains the specified sampling parameters for DAQ. The sampling parameters specified must not exceed the capability of the ADC device.

TriggerParameters

Address of a TriggerParametersStruct structure that contains the specified trigger parameters for DAQ. The trigger parameters specified must not exceed the capability of the ADC device if hardware trigger or external trigger is used. For software trigger, the trigger capacity depends on the interface DLLs. Software Trigger is only possible for those ADC devices that support continuous streaming, such as sound cards.

DAQData

Address of a DAQDataStruct structure.

DAQDAOSyncParameters

Address of a DAQDAOSyncParametersStruct structure. This parameter is used only if you need to synchronize the DAO and DAQ processes, and the synchronization is only possible if software trigger is used in the DAQ. Otherwise, it should be set to NULL.

dwCallback

Address of a handle to a window, or the identifier of a thread to be called during DAQ to process messages related to the progress of DAQ.

fdwOpen

- 0: *dwCallback* is a window handle.
- 1: *dwCallback* is a thread identifier.

Return Values

Reserved.

2.2.2 DAQ_Start

The DAQ_Start function starts the DAQ process. It should be called after DAQ_SetParameters.

```
int DAQ_Start()
```

Return Values

- 0: Successful
- 1: Fail to start DAQ
- 2: Sampling frequency not supported
- 3: Buffer size exceeded.
- 4: DAQ card not found
- 5: Trigger Delay Percentage not supported
- 6: ADC Range not supported

2.2.3 DAQ_Stop

The DAQ_Stop function stops the DAQ process.

```
int DAQ_Stop()
```

Return Values

Reserved.

2.2.4 DAQ_GetSamplePosition

The DAQ_GetSamplePosition function retrieves the current input position.

```
int DAQ_GetSamplePosition()
```

Return Values

Sample No..

2.2.5 DAQ_GetDeviceList

The DAQ_GetDeviceList function retrieves a list of the ADC devices of the same category present in the system. It may also be used to retrieve a list of channels for a specified device. You may use the retrieved information to determine which device or which channel to use for DAQ.

```
int DAQ_GetDeviceList(  
char **ppDevList,  
int MaxEntries,  
int MaxLength,  
int DeviceNo  
);
```

Parameters

ppDevList

Pointer to a string array. Each string will contain a device name. It is NULL terminated.

MaxEntries

The maximum number of the strings allocated by the calling program.

MaxLength

The maximum length of each string allocated by the calling program.

DeviceNo

-1: to get a list of device names.

>=0: Device No., to get a list of channel names for the specified device. (applicable for SoundCardASIODAQO.dll)

Return Values

Number of Devices or number of channels.

2.2.6 DAQ_GetDeviceInfo

The DAQ_GetDeviceInfo function retrieves the information of a specified ADC device present in the system. You may use the retrieved information to determine the sampling and trigger capacity of the device.

```
DAQ_GetDeviceInfo(  
DAQDeviceInfoStruct& DAQDeviceInfo,  
WORD DeviceNo  
);
```

Parameters

DAQDeviceInfo

Address of a DAQDeviceInfo structure.

DeviceNo

Device No. of the device whose information to be retrieved.

Return Values

Reserved.

2.2.7 DAQ_Unlock

The DAQ_Unlock function unlocks the interface DLL so that its functions can be used by the calling program. This function must be called before any interface functions 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 interface DLL.

nSerialNumberPart1

Part 2 of the serial number of the interface DLL.

nSerialNumberPart1

Part 3 of the serial number of the interface DLL.

nSerialNumberPart1

Part 4 of the serial number of the interface DLL.

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 vtDAQ DLLs, such as the trial version, the softkey activated version, the USB hardkey activated version and the DSO hardware bundled version, a generic serial number 0000-0000-0000-0000 should be used. Note that for the trial version and the softkey activated version, a warning message will pop up showing that the DLL is a trial version. The message will not show up if a USB hardkey or any VT DSO hardware is connected to your computer.
3. For not-copy-protected vtDAQ DLLs, which is usually the case for OEM, a customer specific serial number will be given when the DLL is purchased from Virtins Technology.

2.2.8 DAQ_Load

Reserved.

2.2.9 DAQ_Unload

Reserved.

2.3 Messages and Status Flags

2.3.1 WM_MYMESSAGE_DAQ_START

This message is sent when the device is started using DAQ_Start. Meanwhile, the second bit of Status in DAQDataStruct is set.

2.3.2 WM_MYMESSAGE_DAQ_DATA

This message is sent when the device has recorded one frame of data, i.e. one record buffer is full. Meanwhile, the RecordDataCount in DAQDataStruct will be changed from zero to the actual bytes that have been recorded in that record buffer.

The wParam parameter of this message will be filled with the buffer no. of the returned record buffer.

2.3.3 WM_MYMESSAGE_DAQ_STOP

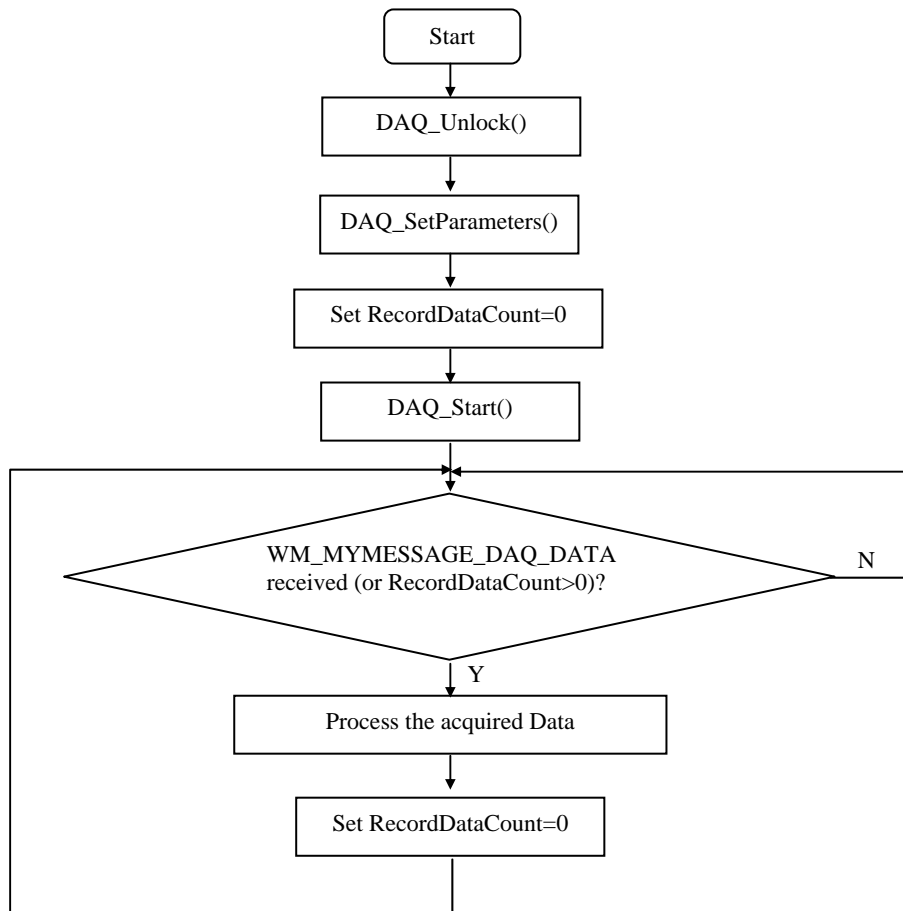
This message is sent when the device is stopped using DAQ_Stop. Meanwhile, the second bit of Status in DAQDataStruct is reset.

2.3.4 WM_MYMESSAGE_DAQ_ERROR

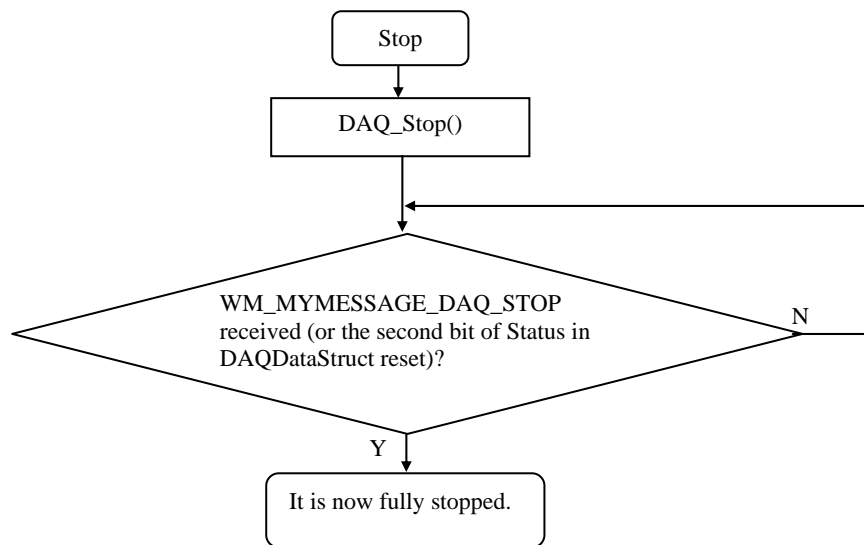
This message is sent when the device has encountered errors.

3. vtDAQ Development Guide

3.1 Flowcharts



Start DAQ



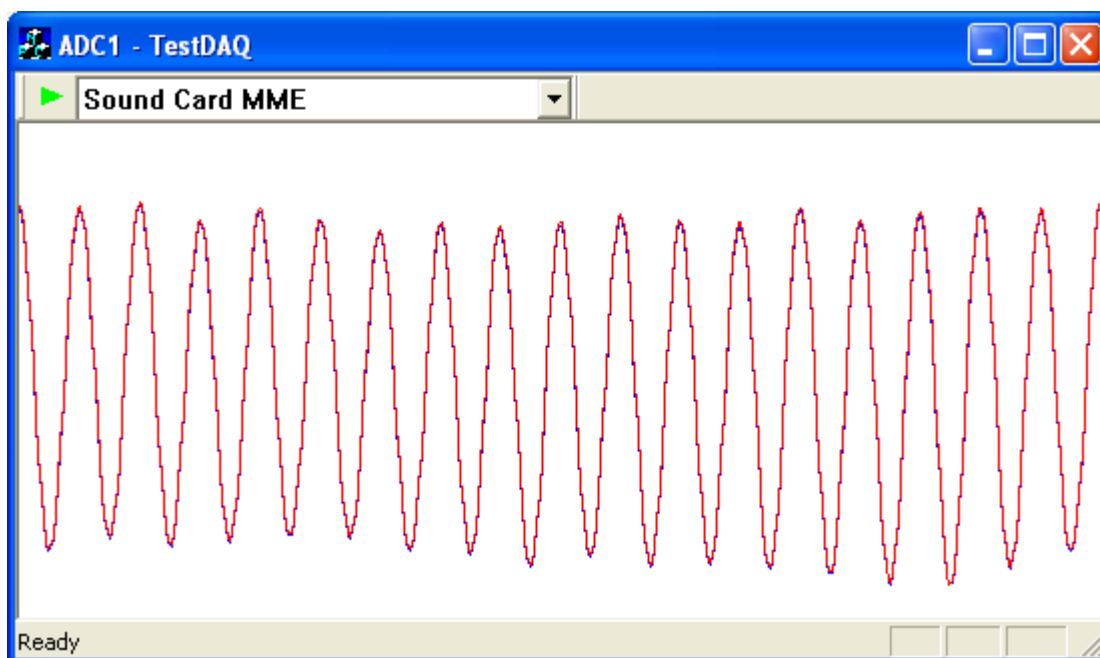
Stop DAQ

3.2 Basic Files

1. Header file to be included: VirtinsDAQ.h
2. vtDAQ interface DLLs:
 - (1) SoundCardMMEDAQ.dll for sound card MME driver
 - (2) SoundCardASIODAQO.dll for sound card ASIO driver.
 - (3) NIDAQ.dll for NI DAQmx compatible cards.
 - (4) VTDSOH1.dll and VTDSOH1drv.dll for VT DSO H1 devices.
 - (5) VTDSOH2.dll and VTDSOH2drv.dll for VT DSO H2 devices.
 - (6) VTDSOF1.dll and VTDSOF1drv.dll for VT DSO F1 devices.
 - (7) VTDSOH3.dll and VTDSOH3drv.dll for VT DSO H3 devices.
 - (8) Any other vtDAQ compatible DLLs.

4. Sample Programs

4.1 TestDAQ written in Visual C++ 6.0



TestDAQ is a sample DAQ back-end program. It demonstrates how to use the vtDAQ interfaces to perform data acquisition. As shown in the above screenshot, there are one Start/Stop button for starting/stopping DAQ, and one combo box for selecting vtDAQ interface DLLs. All other DAQ parameters are set inside the software codes. No GUIs are provided for changing these parameters for simplicity purpose. The program also demonstrates how easily a back-end program can interface to a variety of vtDAQ compatible devices, currently including:

- Sound Cards (MME)
- Sound Cards (ASIO)
- NI DAQmx Cards
- VT DSO H1
- VT DSO H2
- VT DSO H3
- VT DSO F1
- My DAQ Device

4.2 MyDAQ.dll written in Visual C++ 6.0

MyDAQ.dll is a sample DAQ intermediate interface DLL. It demonstrates how to program an intermediate interface DLL that conforms to vtDAQ interface specifications, in order to allow Multi-Instrument to interface to a proprietary DAQ device. The sample codes define a virtual DAQ device with the following properties through `DAQ_GetDeviceInfo()`:

- Two Input Channels with selectable bit depth: 8, 16, 24.
- 100MHz or 50MHz simultaneous sampling rate per channel.
- Voltage Range: $\pm 1V$, $\pm 2V$,
- Coupling Types: AC, DC
- Buffer Size: 20000 bytes per channel
- Hardware trigger supported
- Hardware trigger level adjustable
- Hardware pre-trigger supported

To connect this DLL with Multi-Instrument, you need to put it under the root directory of Multi-Instrument, launch Multi-Instrument, stop the oscilloscope if it is running, and then go to [Setting]>[ADC Device Database], select “My DAQ Device” under Device Category. All the parameters of My DAQ device will be loaded into the ADC database editor. You can modify these parameters if necessary, and then press “Add”. The device is then added into Multi-Instrument’s ADC device database. Now, you can use this device by going to [Setting]>[ADC Device] and select it from the “Device Model” field.

This sample DLL is programmed to support:

- Sampling rates: 100MHz, 50MHz
- Trigger Mode: Auto (Free Run), Normal, Single, or Slow
- Trigger Source: A or B
- Trigger Edge: Up and Down
- Trigger Level: -100%~100%
- Trigger Delay: -100%~100%

The sample DLL does not connected to any physical DAQ device. Instead, it has a built-in simulation data generator, which generates 1Vpp 100kHz sinewave for Channel A and 0.5Vpp 50kHz sinewave for Channel B. The triggering function is also simulated in the DLL.

5. vtDAO Interface Specifications

5.1 Structure Definition

5.1.1 OutputSamplingParametersStruct

```
struct OutputSamplingParametersStruct
{
    double SamplingFrequency;
    WORD SamplingChannels;
    WORD SamplingBitResolution;
    DWORD BufferLength;
    WORD DeviceNo;
    WORD ChannelNo[32];
    double HighLimit[32];
    double LowLimit[32];
    int Mode;
    double Duration;
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```

Members

SamplingFrequency
Sampling Frequency in Hz.

SamplingChannels
Number of Sampling Channels.

SamplingBitResolution
Bit resolution of the DAO data. It can only be 8, 16, 24, or 32 bits. It should generally be equal to the bit resolution of the DAC device. However, if the bit resolution of the DAC device is not an integer multiple of 8, then an integer multiple of 8 nearest to but greater than the bit resolution of the DAC device should be used. It is the intermediate DLL's responsibility to convert the bit resolution of the DAO data to the bit resolution of the DAC device. For example, NI USB-6009 has a bit resolution of 12, this parameter should then be set to 16, and the NIDAO.dll will convert the 16-bit data to 12-bit data and send them to the DAC device.

BufferLength
Buffer Length. It should not exceed the buffer size of the device. For devices that support continuous data streaming or software timed sampling clock, the buffer size can be considered as unlimited.

DeviceNo
Device No. of the same category of DAC devices present in the system. If there is only one such device in the system, then its value should be zero. If there are multiple such devices in the system, then this parameter specifies

which one to use. For example, for SoundCardMMEDAO.dll, it specifies which sound card to use under Windows OS before Windows Vista; It specifies which endpoint (i.e. which output destination (e.g. speaker) of which sound card) to use under Windows Vista.

ChannelNo[32]

This array assigns each sampling channel with a physical channel No.. The sampling channel numbers must start from 0 to *SamplingChannels-1*, and each sampling channel must be assigned with a physical channel No.. A physical channel is a channel in the DAC device. For example, if the DAC device supports 16 channels, and you want to sample only Channel 5 and Channel 9 out of the 16 channels, then you should specify:

ChannelNo[0] = 5
ChannelNo[1] = 9

These parameters are not used by SoundCardMMEDAQ.dll.

For SoundCardASIODAQO.dll, ChannelNo[16] is the physical channel No. of the DAQ input channel A, and ChannelNo[17] is the physical channel No. of the DAQ input channel B.

HighLimit[32]

This array specifies the DAC high limit of each sampling channel.

These parameters are not used by SoundCardMMEDAO.dll and SoundCardASIODAQO.dll.

LowLimit[32]

This array specifies the DAC low limit of each sampling channel.

These parameters are not used by SoundCardMMEDAO.dll and SoundCardASIODAQO.dll.

Mode

For hardware sampling clock:

- 1: require new data every second, run for specified seconds and then auto stop
- 0: require new data every second, forever until manual stop
- 1: do not require new data, run for specified seconds and then auto stop (write once)
- 2: do not require new data, run forever until manual stop (write once)

For software timed sampling clock:

- 9: require new data every second, run for specified seconds and then auto stop
- 10: require new data every second, forever until manual stop
- 11: do not require new data, run for specified seconds and then auto stop (write once)
- 12: do not require new data, run forever until manual stop (write once)

Duration

Signal output duration in second.

ReservedDouble[8]

Reserved.

ReservedDWORD[8]

Reserved.

5.1.2 DAODataStruct

```
struct DAODataStruct
{
    char * pData;
    void (* DataNotify)(BOOL PrepareDataFlag, BOOL
                        NotifyFlag);
    WORD Status;
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```

Members

pData

Address of a “char” data array. The “char” data array is called the output buffer. It contains one second’s output data, i.e. the size of the output buffer is equal to [Sampling Frequency] × [Sampling Channels] × [Sampling Bit Resolution] / 8. Note that for 8-bit data, the data are stored in the output buffer as unsigned values; for 16-bit, 24-bit, and 32-bit data, the data are stored in the output buffer as signed values. This conforms to the data format in a wave file.

DataNotify(BOOL PrepareDataFlag, BOOL NotifyFlag)

Address of a fixed callback function to be called during the progress of data output. The *DataNotify* function is a placeholder for the application-defined function name. It has two parameters:

PrepareDataFlag

TRUE: New data to be prepared

FALSE: No need to prepare new data

NotifyFlag

TRUE: One output buffer has just completed data output.

FALSE: No output buffer has completed data output. This is the case during output buffer initialization. This callback function will be called with this flag set to FALSE once for mode = 1, 2, 11 or 12 and twice (i.e. double buffering) for mode=-1, 0, 9 or 10.

Status

DAO status
Bit0: Not used
Bit1: 0: Stop; 1: Running

ReservedDouble[8]
Reserved.

ReservedDWORD[8]
Reserved.

5.1.3 DAODeviceInfoStruct

```
struct DAODeviceInfoStruct
{
    char ProductCategory[255];
    char ProductType[255];
    char ProductNumber[255];
    char DeviceSerialNumber[255];
    char ChassisModuleName[255];
    BOOL AnalogTriggerSupported;
    BOOL DigitalTriggerSupported;
    WORD NumberOfAOs;
    double MaximumRate;
    double MinimumRate;
    BOOL SampleClockSupported;
    double VoltageRange[64];
    double FrequencyRange[64];
    WORD BitRange[32];
    DWORD BufferSize;
    BOOL Validity;
    double ReservedDouble[8];
    DWORD ReservedDWORD[8];
};
```

Members

ProductCategory[255]
Product Category name.

ProductType[255]
Product Type name.

ProductNumber[255]
Product Number.

DeviceSerialNumber[255]
Device Serial Number.

ChassisModuleName[255]

Chassis Module Name.

AnalogTriggerSupported

Indicates if the device supports analog trigger.

FALSE: Not supported.

YES: Supported.

DigitalTriggerSupported

Indicates if the device supports digital trigger.

FALSE: Not supported.

YES: Supported.

HardwareTriggerLevelAdjustable

Indicates if the hardware trigger level is adjustable.

FALSE: Not supported.

YES: Supported.

NumberOfAOs

Indicates the number of AO output channels of the device.

MaximumRate

Indicates the maximum sampling rate of a channel of the device.

MinimumRate

Indicates the minimum sampling rate of a channel of the device.

SampleClockSupported

Indicates if the device supports hardware sampling clock. If hardware sampling clock is not supported, software timed clock can be used.

VoltageRange[64]

Indicates the pairs of output voltage ranges supported by the device. Each pair consists of a low limit, followed by a high limit. The pairs are stored in ascending order. The not-used elements of this array will be filled with zeros.

FrequencyRange[64]

Indicates the pairs of output sampling frequency ranges supported by this device. Each pair consists of a low limit, followed by a high limit. The pairs are stored in ascending order. The not-used elements of this array will be filled with zeros.

BitRange[32]

Indicates the pairs of output bit resolutions supported by the device. The values are stored in ascending order. The not-used elements of this array will be filled with zeros.

BufferSize

Indicates the buffer size (in samples) per channel. A value of 4294967295 indicates that there is no limit on the buffer size. A value of zero indicates that the hardware supports software timed sampling clock only.

Validity

Reserved.

ReservedDouble[8]

Reserved.

ReservedDWORD[8]

Reserved.

5.2 APIs

5.2.1 DAO_SetParameters

The DAO_SetParameters function sets the DAO parameters.

```
int DAQ_SetParameters(
OutputSamplingParametersStruct& OutputSamplingParameters,
DAODataStruct& DAOData,
DWORD dwCallBack,
DWORD fdwOpen
);
```

Parameters

OutputSamplingParameters

Address of an OutputSamplingParametersStruct structure that contains the specified sampling parameters for DAO. The sampling parameters specified must not exceed the capability of the DAC device.

DAOData

Address of a DAODataStruct structure.

dwCallBack

Address of a handle to a window, or the identifier of a thread to be called during DAO to process messages related to the progress of DAO.

fdwOpen

0: *dwCallBack* is a window handle.
1: *dwCallBack* is a thread identifier.

Return Values

Reserved.

5.2.2 DAO_Start

The `DAO_Start` function starts the DAO process. It should be called after `DAO_SetParameters`.

```
int DAO_Start()
```

Return Values

- 0: Successful
- 1: Fail to start DAO
- 2: Sampling frequency not supported
- 3: Buffer size exceeded.
- 4: DAO card not found

5.2.3 DAO_Stop

The `DAO_Stop` function stops the DAO process.

```
int DAO_Stop()
```

Return Values

Reserved.

5.2.4 DAO_GetSamplePosition

The `DAO_GetSamplePosition` function retrieves the current output position.

```
int DAO_GetSamplePosition()
```

Return Values

Sample No..

5.2.5 DAO_GetDeviceList

The `DAO_GetDeviceList` function retrieves a list of the DAC devices of the same category present in the system. It may also be used to retrieve a list of channels for a specified device. You may use the retrieved information to determine which device or which channel to use for DAO.

```
int DAO_GetDeviceList(  
char **ppDevList,  
int MaxEntries,  
int MaxLength,  
int DeviceNo  
);
```

Parameters

ppDevList

Pointer to a string array. Each string will contain a device name. It is NULL terminated.

MaxEntries

The maximum number of the strings allocated by the calling program.

MaxLength

The maximum length of each string allocated by the calling program.

DeviceNo

-1: to get a list of device names.

>=0: Device No., to get a list of channel names for the specified device. (applicable for SoundCardASIODAQO.dll)

Return Values

Number of Devices or number of channels.

5.2.6 DAO_GetDeviceInfo

The DAO_GetDeviceInfo function retrieves the information of a specified DAC device present in the system. You may use the retrieved information to determine the sampling capacity of the device.

```
DAO_GetDeviceInfo(
DAODeviceInfoStruct& DAODeviceInfo,
WORD DeviceNo
);
```

Parameters

DAODeviceInfo

Address of a DAODeviceInfo structure.

DeviceNo

Device No. of the device whose information to be retrieved.

Return Values

Reserved.

5.2.7 DAO_Unlock

The DAO_Unlock function unlocks the interface DLL so that it can be used by the calling program. This function must be called before any interface functions 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 interface DLL.

nSerialNumberPart1

Part 2 of the serial number of the interface DLL.

nSerialNumberPart1

Part 3 of the serial number of the interface DLL.

nSerialNumberPart1

Part 4 of the serial number of the interface DLL.

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 vtDAO DLLs, such as the trial version, the softkey activated version, the USB hardkey activated version and the DAO hardware bundled version, a generic serial number 0000-0000-0000-0000 should be used. Note that for the trial version and the softkey activated version, a warning message will pop up showing that the DLL is a trial version. The message will not show up if a USB hardkey or any VT DAO hardware is connected to your computer.
3. For not-copy-protected vtDAO DLLs, which is usually the case for OEM, a customer specific serial number will be given when the DLL is purchased from Virtins Technology.

5.2.8 DAO_Load

Reserved.

5.2.9 DAO_Unload

Reserved.

5.2.10 DAO_Write

Reserved.

5.3 Messages and Status Flags

5.3.1 WM_MYMESSAGE_DAO_START

This message is sent when the device is started using DAO_Start. Meanwhile, the second bit of Status in DAODataStruct is set.

5.3.2 WM_MYMESSAGE_DAO_DATA

This message is sent when the output buffer is being returned to the calling program. The buffer is returned to the calling program when it has just been output.

5.3.3 WM_MYMESSAGE_DAO_STOP

This message is sent when the device is stopped using DAO_Stop. Meanwhile, the second bit of Status in DAODataStruct is reset.

5.3.4 WM_MYMESSAGE_DAO_ERROR

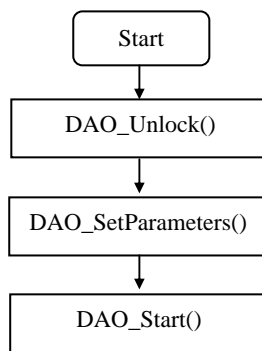
This message is sent when the device has encountered errors.

5.3.5 WM_MYMESSAGE_DAO_STOP_REQUEST

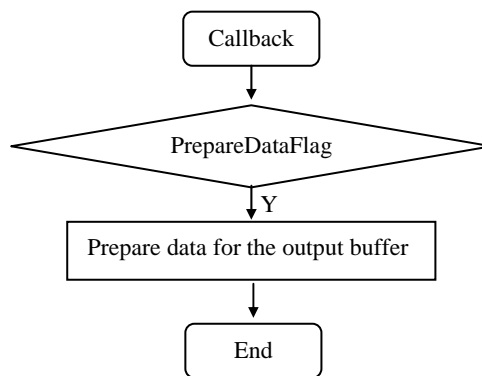
This message is sent when the interface DLL requests the calling program to stop DAO. Upon receiving this message, the calling program should execute the DAO_Stop command.

6. vtDAO Development Guide

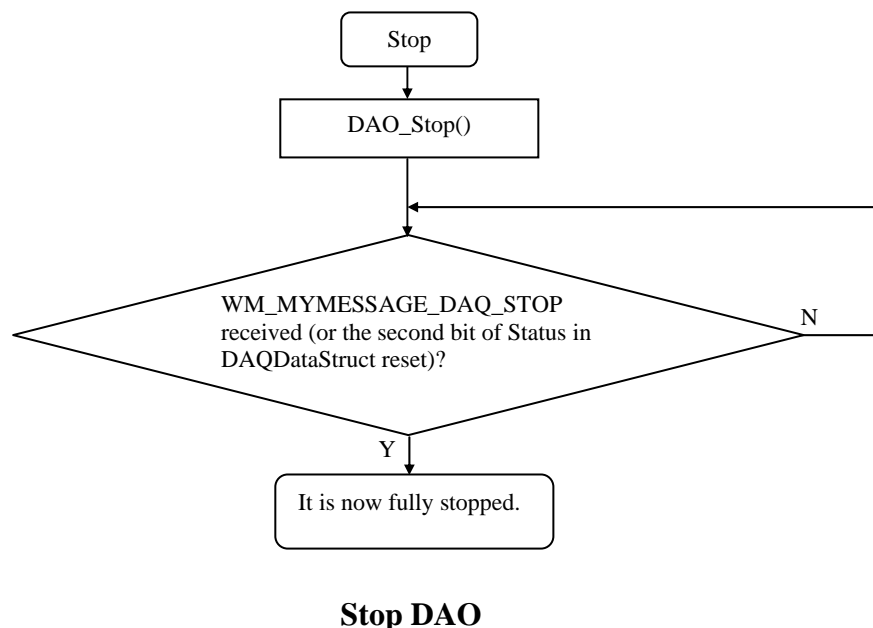
6.1 Flowcharts



Start DAO



Callback Function DataNotify



6.2 Basic Files

1. Header file to be included: VirtinsDAO.h
2. vtDAO interface DLLs:
 - (1) SoundCardMMEDAO.dll for sound card MME driver
 - (2) SoundCardASIODAQO.dll for sound card ASIO driver.
 - (3) NIDAO.dll for NI DAQmx compatible cards.
 - (4) Any other vtDAO compatible DLLs.

6.3 How to Choose Correct Output Mode

6.3.1 Hardware Sampling Clock

If the data to be output repeats every second, then mode 1 or 2 should be used. If the data output to be stopped automatically by the interface DLL at the end of the specified duration, then mode 1 should be used. If mode 2 is used, the data output will not stop until the calling program calls `DAO_Stop`.

If the data to be output changes every second, then mode -1 or 0 should be used. The calling program should prepare new data whenever the callback function `DataNotify` is called and its `PrepareDataFlag` is set. If the data output to be stopped automatically by the interface DLL at the end of the specified duration, then mode -1 should be used. If mode 0 is used, the data output will not stop until the calling program calls `DAO_Stop`.

6.3.2 Software Timed Sampling Clock

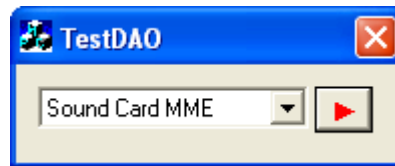
If the data to be output repeats every second, then mode 11 or 12 should be used. If the data output to be stopped automatically by the interface DLL at the end of the specified duration, then mode 11 should be used. If mode 12 is used, the data output will not stop until the calling program calls `DAO_Stop`.

If the data to be output changes every second, then mode 9 or 10 should be used. The calling program should prepare new data whenever the callback function `DataNotify` is called and its `PrepareDataFlag` is set. If the data output to be stopped automatically by the interface DLL at the end of the specified duration, then mode 9 should be used. If mode 10 is used, the data output will not stop until the calling program calls `DAO_Stop`.

Generally, software timed sampling clock is not so accurate as hardware sampling clock. The timing task is performed by the interface DLL and its accuracy depends on the current workload of the computer system.

7. Sample Programs

7.1 TestDAO written in Visual C++ 6.0



TestDAO is a sample DAO back-end program. It demonstrates how to use the vtDAO interfaces to perform data output. There are one Start/Stop button for starting/stopping DAO and one combo box for selecting vtDAO interface DLLs. All the DAO parameters are set inside the software codes. No GUIs are provided for changing these parameters for simplicity purpose.