vtDAQ and vtDAO APIs

Version 1.9

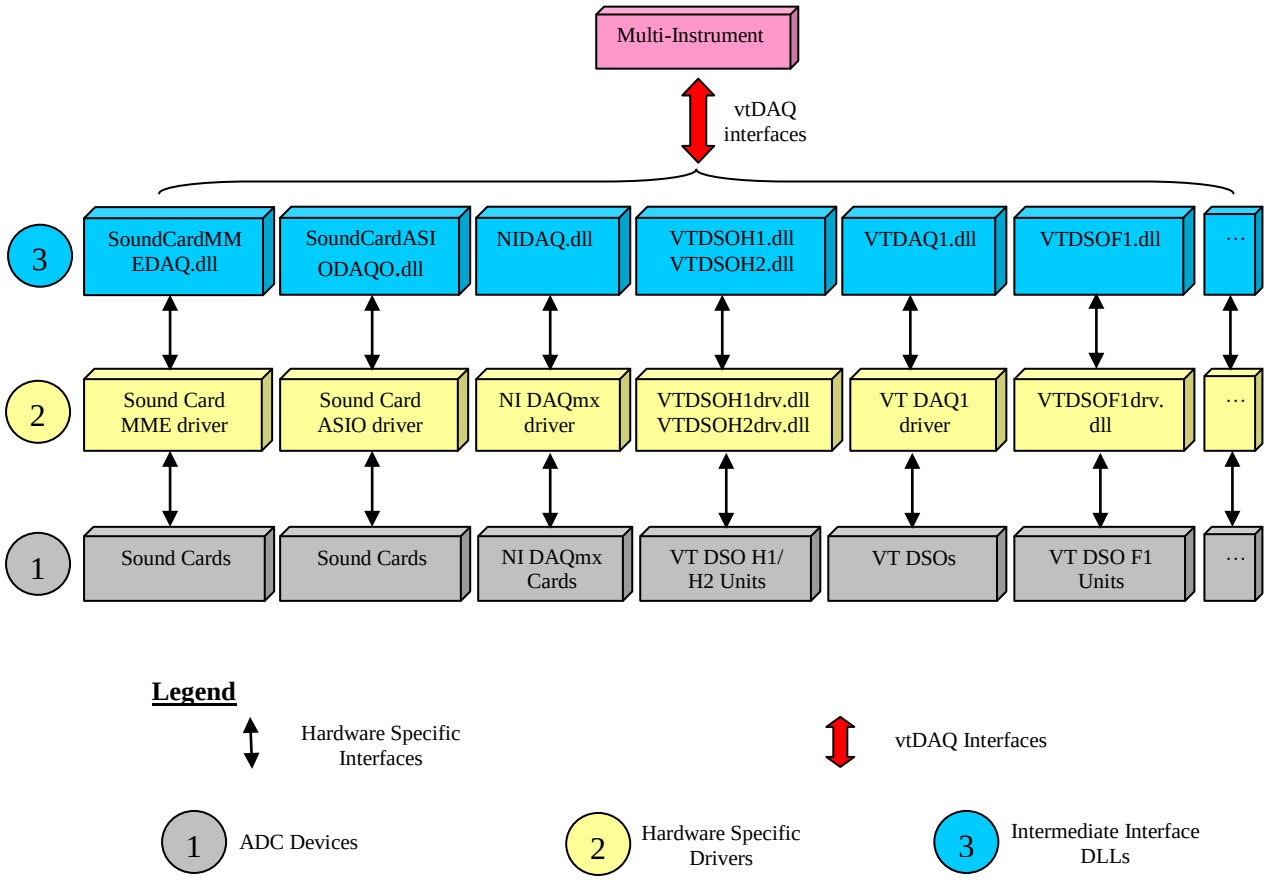
Note: VIRTINS TECHNOLOGY reserves the right to make modifications to this document at any time without notice. This document may contain typographical errors.

TABLE OF CONTENTS

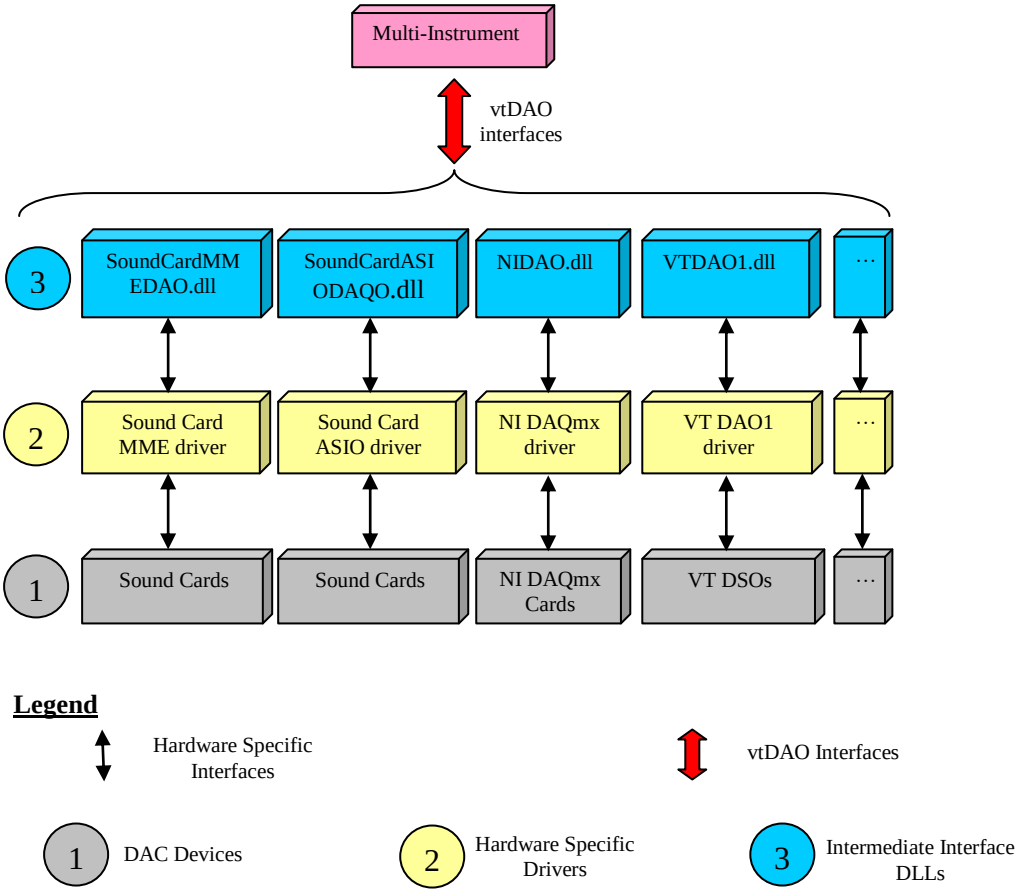
1. Introduction

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.



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. The record buffer size (in bytes) is determined by [Record Length] × [Sampling Channels] × [Sampling Bit Resolution] / 8. This size should not exceed the memory size of the device. For devices that support continuous data streaming such as a sound card, the device memory size can be considered as unlimited. For VT DSO under Record Mode (a continuous data streaming mode, e.g. the Record Mode and Roll Mode in Multi-Instrument), the record buffer size should not exceed ½ of the memory size of the device due to the double buffering mechanism implemented in the hardware.

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 later versions.

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: Non referenced single-ended mode
3: Differential mode
4: Pseudo differential mode

These parameters are not used by SoundCardMMEDAQ.dll, SoundCardASIODAQO.dll, VTDSOH1.dll, VTDSOH2.dll, VTDSOF1.dll, and VTDAQ1.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:
0: Auto; 1: Max; 2: Min.
ReservedDWORD[3]-BIT0: 0: Interleaved 1: Channel by Channel

For SoundCardDAQ.dll, SoundCardASIODAQO.dll and NIDAQ.dll,
[2]-BIT1: Enable High Frequency Rejection for Trigger Frequency Rejection HNX option

For Second-Generation VT DSOs:
[1]: Number of Digital Channels
[2]-BIT0: Enable BitResolutionEnhancement
[2]-BIT1: Enable High Frequency Rejection for Trigger Frequency Rejection HNX option
[3]-BIT0: 0: Interleaved 1: Channel by Channel

For SoundCardDAQ.dll and SoundCardASIODAQO.dll, ReservedDWORD[4] is used to differentiate Device Models:

ReservedDWORD[4]~BIT0~BIT7 for Main Device Models:
0: Sound Cards
1: IEPE-2G05
2: CAMP-2G05
3: RTA-1G05
100: RTX6001
101: RME ADI-2 Pro
102: RME ADI-2/4 PRO SE

ReservedDWORD[4]~BIT8~BIT15 for Sub Device Models:
0: IEPE-2G05
1: IEPE-2G05A
2: IEPE-2G05B
3: IEPE-2G05C
5: IEPE-2G05D
6: IEPE-2G05E

For IEPE-2G05, CAMP-2G05, RTA-1G05:

[5]-BIT0~BIT2: Ch. A High Pass Filter
0: None 1: 1.8Hz 2: 119Hz 3: 236Hz 4: 464Hz

[5]-BIT3~BIT5: Ch. B High Pass Filter
0: None 1: 1.8Hz 2: 119Hz 3: 236Hz 4: 464Hz

[5]-BIT6~BIT7: Input & Output wiring
0: None 1: oA←iA, oB←iB 2: iB<-oA

[5]-BIT8: CAMP-2G05 reset-on-start flag

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.

HardwareTriggeres

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.

For SoundCardDAQ.dll, SoundCardASIODAQO.dll and NIDAQ.dll,
[2]: Noise Rejection Hysteresis Percentage for Trigger Frequency Rejection HNX option (e.g.10 for 10%)

For Second-Generation VT DSOs,
[1]: External Trigger Level Percentage
[2]: Noise Rejection Hysteresis Percentage for Trigger Frequency Rejection HNX option (e.g.10 for 10%)

ReservedDWORD[8]

Reserved.
For Second-Generation VT DSOs, SoundCardDAQ.dll, NIDAQ.dll, and SoundCardASIODAQO.dll,
[0]: Trigger Frequency Rejection
0: NIL
1: HFR (High Frequency Rejection)
2: NR0 (Noise Rejection Level 0)
3: NR1; 4: NR2; 5: NR3; 6: NR4;
7: HN0 (High Frequency Rejection + Noise Rejection Level 0);
8: HN1; 9: HN2; 10: HN3; 11: HN4;
12: HNX (High Frequency Rejection and / or Noise Rejection Level X)

For Second-Generation VT DSOs,
[1]: Trigger Slave / Master
0: Slave 1: Master
[2]: Roll Mode
0: Not Roll Mode 1: Roll Mode

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 (in bytes) 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 in 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 (e.g. Normal Frame Mode), one record buffer is enough. In some special cases (e.g. Record Mode), 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 right before the trigger level is crossed. This parameter is only valid if software trigger or some VT DSO models are used. The value is stored as an unsigned value.

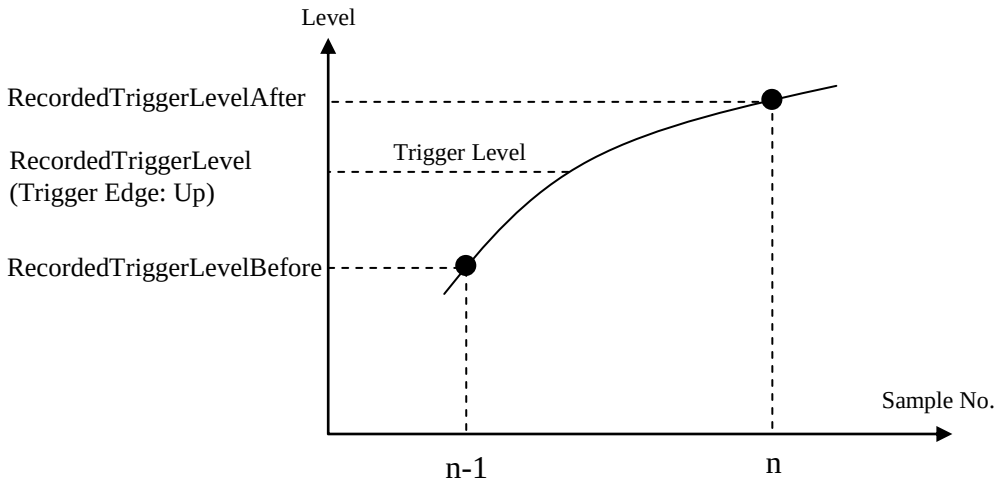
RecordedTriggerLevelAfter

The value of the sample right after the trigger level is crossed. This parameter is only valid if software trigger or some VT DSO models are used. The value is stored as an unsigned value.

RecordedTriggerLevel

Trigger level. This parameter is only valid if software trigger or some VT DSO models are 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.

For Second-Generation VT DSOs, SoundCardDAQ.dll, NIDAQ.dll, and SoundCardASIODAQO.dll:

[0]: The value of the sample right before the first recorded sample in the current data frame in Ch.A. This parameter is only valid if software trigger or some VT DSO models are used. The value is stored as an unsigned value. It is equal to *RecordedTriggerLevelBefore* only when the *TriggerDelayPercent* in the Trigger Parameters is 0% and the *Trigger Source* is Ch.A.

[1]: The value of the sample right before the first recorded sample in the current data frame in Ch.B. This parameter is only valid if software trigger or some VT DSO models are used. The value is stored as an unsigned value. It is equal to *RecordedTriggerLevelBefore* only when the *TriggerDelayPercent* in the Trigger Parameters is 0% and the *Trigger Source* is Ch.B.

For SoundCardASIODAQO.dll:

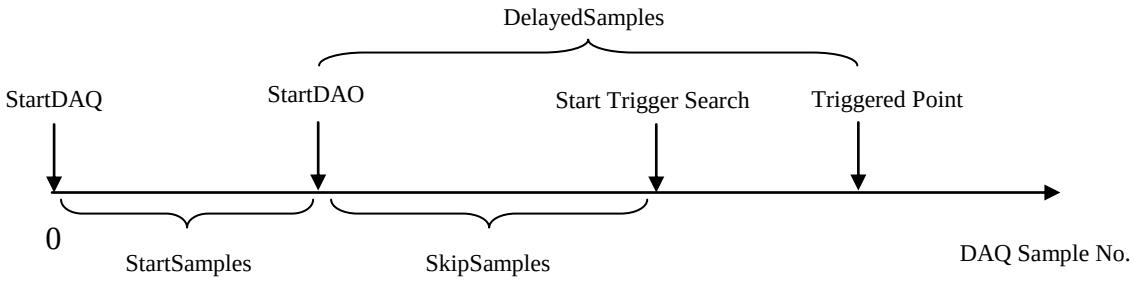
[2]: Data Format

0: integer

1: 32-bit float format

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 ← 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, `MultiChannelRate = SingleChannelRate / 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. If *dwCallBack* =

NULL, then no message will be posted back, that is, the callback function will not be called.

fdwOpen

0: *dwCallBack* is a window handle.
1: *dwCallBack* is a thread identifier.
2: VT hardware initialization and initialize the DLL with the information stored in the VT hardware
(1) For Second-Generation VT DSOs: FPGA download and read calibration data
(2) For IEPE-2G05, CAMP-2G05, RTA-1G05: read calibration data
4: (1) For Second-Generation VT DSOs: initialize the DLL with the information stored in the VT hardware, i.e. read calibration data
(2) For IEPE-2G05, CAMP-2G05, RTA-1G05: (i) Configure High Pass Filter (ii) Configure input & output wiring (iii) Reset CAMP-2G05 once
5: (1) For IEPE-2G05, CAMP-2G05, RTA-1G05: (i) Configure High Pass Filter (ii) Configure input & output wiring (iii) Reset CAMP-2G05 once (iv) Set input voltage ranges without going through DAQ_Start( ), which is convenient for those who use generic sound card APIs such as wave APIs to obtain the raw samples and use vtDAQ API to set the above parameters only.
7: Open the device's own control panel (e.g. ASIO control panel) if any.

Return Values

<0: fail.

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 retrieves 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.

```
int Unlock(
WORD nSerialNumberPart1, //serial number part 1
WORD nSerialNumberPart2, //serial number part 2
WORD nSerialNumberPart3, //serial number part 3
WORD 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.

Return Values

Reserved.

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.

2.4 C++ Wrappers with simple data structures for Labview and others

Some arguments in vtDAQ APIs have a complex data structure which may not be readily supported by other software development tools such as Labview. vtDAQLV.dll is a wrapper developed around all those vtDAQ compatible dlls. It uses simple data structures.

In Labview, vtDAQLV.dll can be called using Call Library function node. Stdcall (WINAPI) calling convention should be used.

For implicit linking in other software tools, the corresponding header file is vtDAQcplus.h and the library file is vtDAQLV.lib.

2.4.1 DAQLV_SetDAQType

The DAQLV_SetDAQType function loads a hardware-specific vtDAQ compatible dll dynamically. This function must be called before any interface functions can be used including the Unlock function.

```
int SetDAQType(
WORD Type
)
```

Parameters

Type

Type of hardware.

0: SoundCardMMEDAQ.dll
1: SoundCardASIODAQO.dll
2: NIDAQ.dll
3: VTDSOH1.dll
4: VTDSOH2.dll
5: VTDSOF1.dll
6: VTDSOH3.dll
7: VTDAQ1.dll
8: Reserved
9: MyDAQ.dll

Return Values

0: Successful.
1: Failed

2.4.2 DAQLV_Unlock

Same as Section 2.2.7.

2.4.3 DAQLV_SetSamplingParameters

Basically it sets the SamplingParameters in DAQ_SetParameters API (Refer to Sections 2.2.1 & 2.1.1).

```
int DAQLV_SetSamplingParameters(
double SamplingFrequency,
WORD SamplingChannels,
WORD SamplingBitResolution,
```



```
DWORD RecordLength,
WORD DeviceNo,
WORD * pChannelNo,
double * pHighLimit,
double * pLowLimit,
WORD * pTerminalConfiguration,
WORD * pCouplingType,
double * pReservedDouble,
DWORD * pReservedDWORD
)
```

Parameters

Refer to Sections 2.1.1.

Return Values

0: Successful.

2.4.4 DAQLV_SetTriggerParameters

Basically it sets the TriggerParameters in DAQ_SetParameters API (Refer to Sections 2.2.1 & 2.1.2).

```
int DAQLV_SetTriggerParameters(
WORD TriggerMode,
WORD TriggerSource,
WORD TriggerEdge,
double TriggerLevelPercent,
double TriggerDelayPercent,
WORD ExtChannelNo,
BYTE RecordMode,
BYTE HardwareTrigger,
double * pReservedDouble,
DWORD * pReservedDWORD
)
```

Parameters

Refer to Sections 2.1.2.

RecordMode:

0: Not Record Mode
1: Record Mode

HardwareTrigger

0: Not Hardware Trigger
1: Hardware Trigger

Return Values

0: Successful.

2.4.5 DAQLV_SetDAQData

Basically it sets the DAQData in DAQ_SetParameters API (Refer to Sections 2.2.1 & 2.1.3).

```
int DAQLV_SetDAQData(
char *ppRecordData,
long * pRecordDataCount,
unsigned long RecordBufferCount,
DWORD *RecordedTriggerLevelBefore,
DWORD *RecordedTriggerLevelAfter,
DWORD *RecordedTriggerLevel,
DWORD *RecordedTimeStampInSeconds,
DWORD *RecordedTimeStampInMilliSeconds,
WORD *Status,
double * pReservedDouble,
DWORD * pReservedDWORD,
DWORD dwCallBack,
DWORD fdwOpen,
double *pCalibratedData,
WORD CalibrationMode
)
```

Parameters

Refer to Sections 2.1.3. There are some important differences to note here.

ppRecordData

A "char" data array. It contains one or multiple record buffers. The size (in bytes) of each record buffer must be equal to [Record Length] × [Sampling Channels] × [Sampling Bit Resolution] / 8. The total size (in bytes) of ppRecordData is then [Record Buffer Count] × [Record Buffer Size].

RecordedTriggerLevelBefore

A pointer to RecordedTriggerLevelBefore.

RecordedTriggerLevelAfter

A pointer to RecordedTriggerLevelAfter.

RecordedTriggerLevel

A pointer to RecordedTriggerLevel.

RecordedTimeStampInSeconds

A pointer to RecordedTimeStamp.time.

RecordedTimeStampInMilliSeconds

A pointer to RecordedTimeStamp.millitm

dwCallBack

Refer to Section 2.2.1.

fdwOpen

Refer to Section 2.2.1. In addition:

0: *dwCallBack* is a window handle, a message will be posted to it whenever a new frame of data is acquired, together with the Record Buffer No. indicated by WPARAM.

1: *dwCallBack* is a thread identifier, a message will be posted to it whenever a new frame of data is acquired, together with the Record Buffer No. indicated by WPARAM.

5: *dwCallBack* is a user event refnum in Labview, a message will be posted to it whenever a new frame of data is acquired, together with the Record Buffer No. indicated by DATA.

pCalibratedData

It contains the calibrated data converted from the raw data in the latest record buffer. The size (in Double) of this CalibratedData array must be equal to [Record Length] × [Sampling Channels].

CalibrationMode

0: Calibrated Data is not needed.
1: Channel by Channel
2: Interleaved

Return Values

0: Successful.

2.4.6 DAQLV_SetParameters

This function is used to perform some special functions in DAQ_SetParameters API when fdwOpen >= 2. DAQLV_SetSamplingParameters, DAQLV_SetTriggerParameters, DAQLV_SetDAQData must be called first.

```
int DAQLV_SetParameters(
DWORD fdwOpen
)
```

Parameters

fdwOpen

fdwOpen >= 2. Refer to Section 2.2.1.

Return Values

0: Successful.

2.4.7 DAQLV_Start

Same as Section 2.2.2, except that it will perform hardware-specific initialization additionally first using DAQ_SetParameters( ) with fdwOpen = 2, if this initialization has not been done using DAQLV_SetParameters( ) with fdwOpen = 2 yet.

2.4.8 DAQLV_Stop

Same as Section 2.2.3.

2.4.9 DAQLV_Acknowledge

After the calling program processes all the recorded data in the record buffer, it must reset the number of recorded bytes (referenced by pRecordDataCount and offset by the record buffer no.) to zero to inform the intermediate DLL that this record buffer is ready to accept new data. DAQLV_Acknowledge can be use for this purpose.

```
int DAQLV_Acknowledge (int nRecordBufferNum)
```

Parameters

nRecordBufferNum: No. of the Record Buffer to be reset.

Return Values

0: Successful.

2.5 C Wrappers with simple data structures for LabWindows/CVI and others

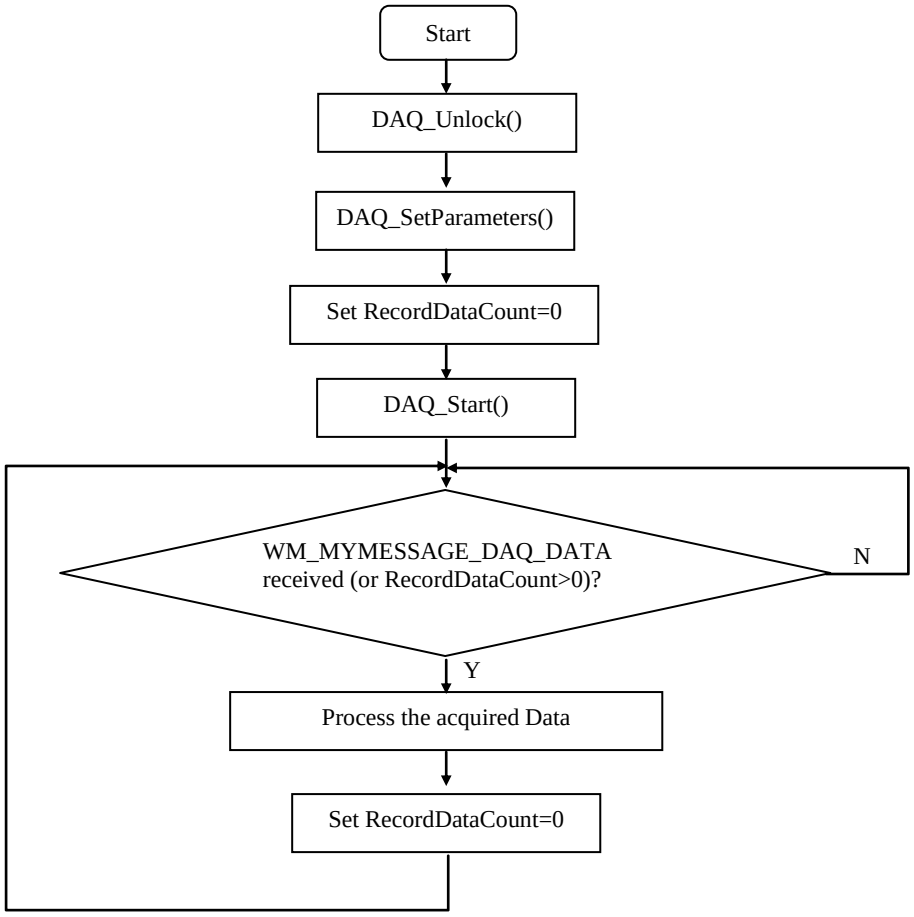
Some software development tools supports ANSI C interfaces only, such as LabWindow/CVI. VtDAQLV.dll provides a set of C interfaces with exactly the same functions and arguments as those described in Section 2.4. They are:

int DAQCVI_SetDAQType
int DAQCVI_Unlock
int DAQCVI_SetSamplingParameters
int DAQCVI_SetTriggerParameters
int DAQCVI_SetDAQData
int DAQCVI_SetParameters
int DAQCVI_Start
int DAQCVI_Stop
int DAQCVI_Acknowledge

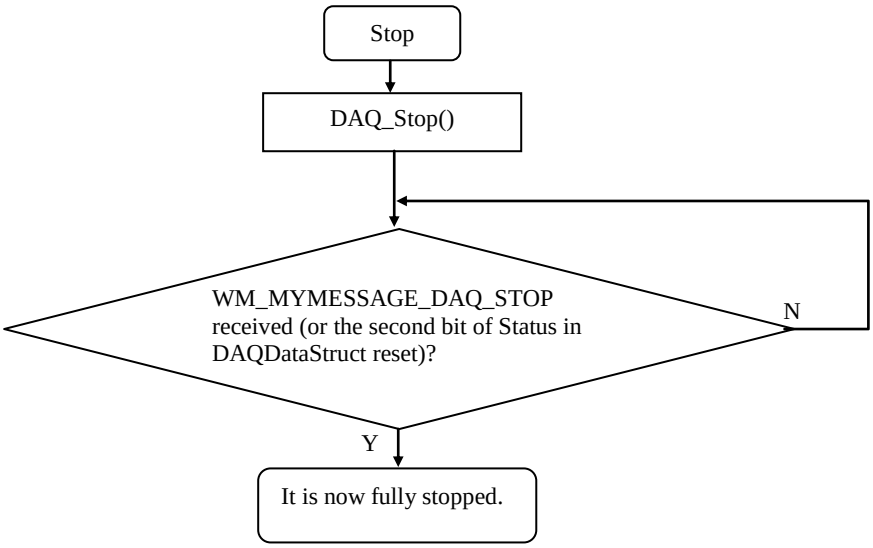
The corresponding header file is vtDAQc.h and the library file is vtDAQLV.lib.

3. vtDAQ Development Guide

3.1 Flowcharts



Start DAQ



Stop DAQ

Please note:

1. Under non Record Mode (i.e. TriggerParameters.RecordMode = FALSE, a.k.a. Normal Frame Mode), there is usually a gap between consecutive data frames. Normally only one record buffer is required (i.e. DAQData.RecordBufferCount = 1). Thus DAQData.ppRecordData contains only one record buffer and DAQData.pRecordDataCount contains only one element. Memory allocation for these two should be done accordingly.

2. Under Record Mode (i.e. TriggerParameters.RecordMode = TRUE), if everything is handled properly (e.g. the sampling rate is not too high, etc.), there will be no gap between consecutive data frames. Multiple record buffers are required (i.e. DAQData.RecordBufferCount >= 2). Thus DAQData.ppRecordData contains multiple record buffers and DAQData.pRecordDataCount contains multiple elements. Memory allocation for these two and handshaking by resetting the corresponding RecordDataCount after the returned record buffer is processed should be done properly.

   Particularly for VT DSOs, the size of each record buffer (in bytes), which is equal to [Record Length] × [Sampling Channels] × [Sampling Bit Resolution] / 8, should not exceed ½ of the DSO memory size due to the double buffering mechanism for the Record Mode implemented in the hardware. A buffer size equal to a multiple of 1024 may improve data transfer efficiency.

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) VTDAQ1.dll for VTDAQ1 devices.
(9) Any other vtDAQ compatible DLLs.

3. vtDAQ interface LIBs:
Every dll comes with its own lib file.

3.3 Auxiliary Files

1. vtDAQ interface DLLs' C++ Wrappers with simple data structures for Labview and others
   (1) Header file to be included: vtDAQcplus.h
   (2) LIB file: vtDAQLV.lib
   (3) DLL file: vtDAQLV.dll (this is in addition to the basic vtDAQ interface DLLs)

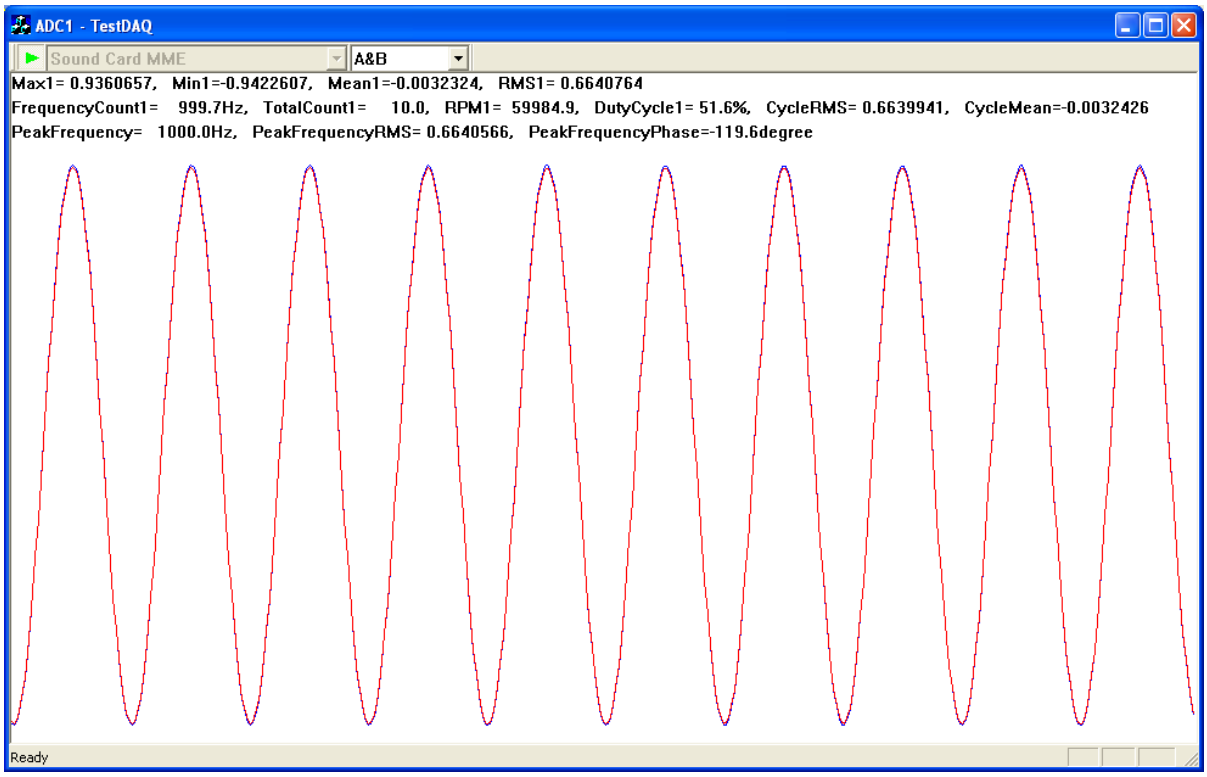
2. vtDAQ interface DLLs' C Wrappers with simple data structures for LabWindows/CVI and others
   (1) Header file to be included: vtDAQc.h
   (2) LIB file: vtDAQLV.lib
   (3) DLL file: vtDAQLV.dll (this is in addition to the basic vtDAQ interface DLLs)

3.4 Hardware Specific Configuration Files

Some hardware devices such as VT DSOs come with some hardware specific configuration files. These files should also be included in the developed software package. For VT DSOs, these files can be found in Multi-Instrument's installation directory\HardwareConfig\….

4. vtDAQ 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, one combo box for selecting vtDAQ interface DLLs, and one combo box for selecting the number of sampling channels. Selection of sampling channels is used to demonstrate how to change a sampling parameter on-the-fly without manually stopping the DAQ first. All other DAQ parameters are set inside the software codes 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
- VT DAQ 1
- My DAQ Device

To facilitate data processing and analysis after data acquisition, Virtins Technology has also developed and exposed a suite of Signal Processing and Analysis APIs (vtSPA). These APIs are also linked inside the TestDAQ program. To demostrate some of the vtSPA features, TestDAQ calculates the Max, Min, Mean, RMS, frequency count, total count, RPM, duty

cycle, cycle RMS, cycle mean, peak frequency, RMS of peak frequency, and phase of peak frequency. These values are displayed in the upper par of the oscilloscope graph.
(please refer to: http://www.virtins.com/Signal-Processing-and-Analysis-APIs.pdf for detailed description of vtSPA).

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: ±1V, ±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.

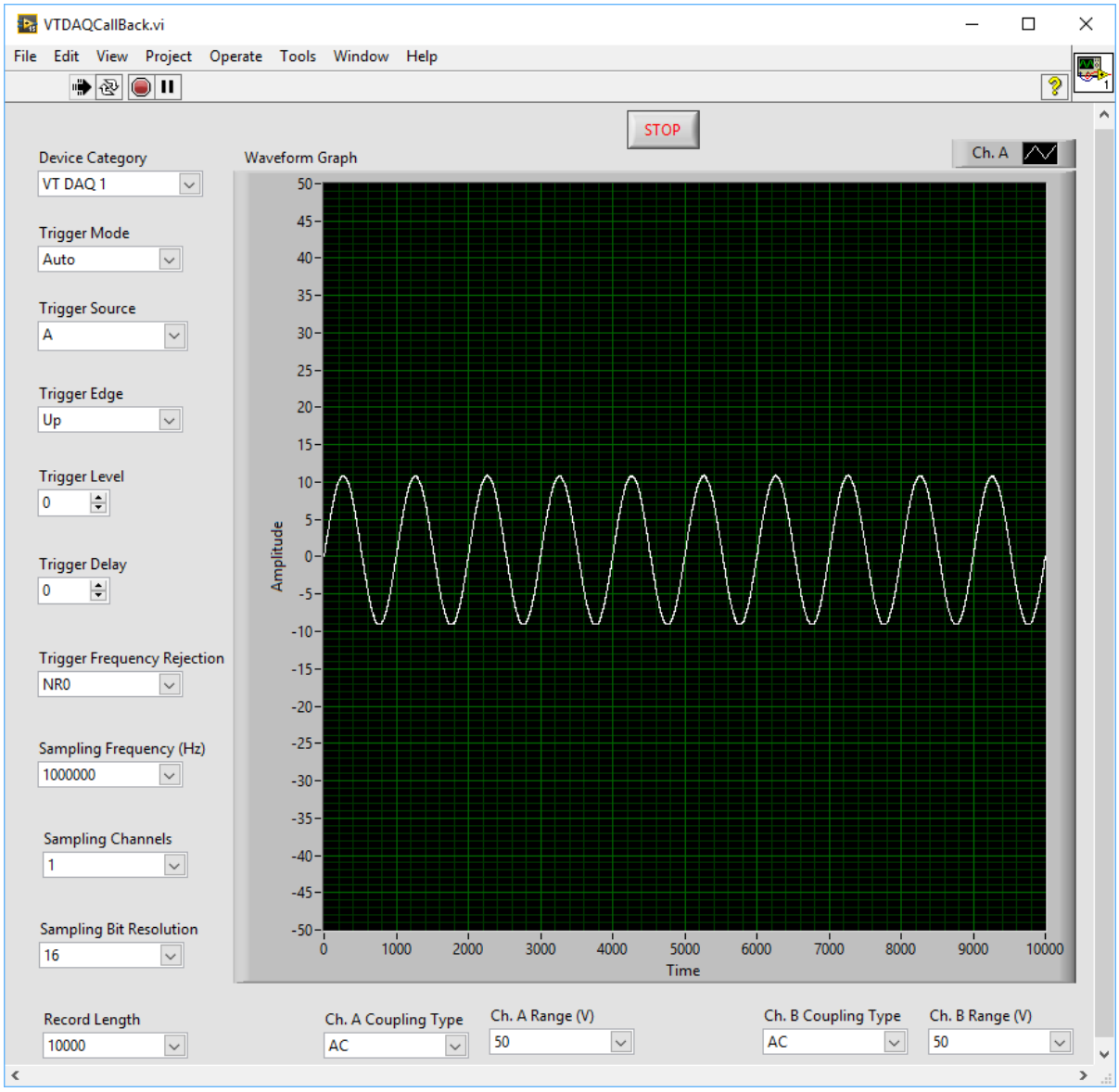
This sample DLL is programmed to support:

- 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.

4.3 vtDAQ Labview Samples

4.3.1 vtDAQCallBack.vi written in Labview 15.0



vtDAQCallBack.vi is a sample VI for Labview programming using vtDAQ interfaces. VtDAQ interface wrapper: vtDAQLV.dll, is called using Labview Call Library function node. It is basically an oscilloscope VI with adjustable sampling and triggering parameters. To keep the VI simple, these parameters cannot be changed on-the-fly.

There are two methods to know if a new frame of data has been acquired and is ready for processing: Callback and Polling. Callback method is faster and more efficient as the vtDAQLV.dll sends notification (a user event) to the VI once a new frame of data is ready, while in Polling method, the VI has to check the "Data Ready" flag at regular intervals to see if the data are ready. This VI uses the callback method. After the data are processed, the VI calls DAQLV_Acknowledge to release the record buffer for new data acquisition.

4.3.2 vtDAQPolling.vi written in Labview 15.0

This VI is similar to vtDAQCallBack.vi. However, it uses Polling method instead.

4.3.3 vtDAQCallBackWithAdjustableNumberOfShots.vi written in Labview 15.0

This VI is similar to vtDAQCallBack.vi. In addition, it is possible to set number of shots. If the specified number of shots is reached, the program will stop automatically. If the number of shots is set to 0, then the program will run forever until the stop button is pressed.

4.3.4 vtDAQCallBackSoundCard.vi written in Labview 15.0

This VI is basically the same as vtDAQCallBack.vi. However, its sampling parameters are set, by default, to values compatible with sound cards. It is easier for those who want to try vtDAQ Labview development without a VT hardware device. It also demonstrates how to take advantage of vtDAQ interface's powerful functions such as software trigger. It should be noted that SLABHIDDevice.dll and SLABHIDtoUART.dll must be put in a directory (e.g. resource, etc.) that is listed in the Labview Call Library Function Node searching paths. Otherwise, "Failed to load SoundCardMMEDAQ.dll" error message will pop up.

4.3.5 vtDAQCallBack_NI-USB-6009.vi written in Labview 15.0

This VI is basically the same as vtDAQCallBack.vi. However, its sampling parameters are set, by default, to values compatible with NI USB-6009. It demonstrates how to take advantage of vtDAQ interface's powerful functions such as software trigger.

4.3.6 vtDAQCallBack_VT-IEPE-2G05.vi written in Labview 15.0

This VI is basically the same as vtDAQCallBack.vi. However, its sampling parameters are set, by default, to values compatible with VT IEPE-2G05. How to configure those VT IEPE-2G05 specific settings such as high-pass filtering and Input & Output wiring are demonstrated. It should be noted that SLABHIDDevice.dll and SLABHIDtoUART.dll must be put in a directory (e.g. resource, etc.) that is listed in the Labview Call Library Function Node searching paths. Otherwise, "Failed to load SoundCardMMEDAQ.dll" error message will pop up.

4.4 TestDAQ written in Visual C# 2012

TestDAQ_CSharp is a sample DAQ back-end program written in Visual C#, with functions similar to its Visual C++ counterpart introduced previously. Instead of calling vtDAQ interface dll directly, it interfaces to vtDAQLV.dll which in turn calls the respective vtDAQ interface dll, thus avoiding using complex data structures in the original vtDAQ APIs. This program also demonstrates how to interface to vtSPA.dll using C#.

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.

For VT DSO:
[0]: DDS Buffer Byte count
[1]-BIT0: EnableTrigger
0: no trigger, output immediately
1: do not output till triggered, trigger conditions set by the oscilloscope trigger
[1]-BIT1: DisableProbeCAL
0: Enable Probe CAL Output
1: Disable Probe CAL Output
[1]-BIT2: EnableDDSInterpolation
0: No Interpolation
1: Interpolation
[1]-BIT3: DDSLoop
0: No loop
1: Loop until manual stop
[1]-BIT4: DDSAmplitudeSweep
0: No Amplitude Sweep
1: Amplitude Sweep
[1]-BIT5: DDSSweepAmplitudeDCSign
0: Positive
1: Negative
[1]-BIT6: ProbeCALWaveform
0: Square
1: MLS
[1]-BIT7: EnableProbeCALFrequencyDivisionRatio
0: use default 1kHz
1: use ProbeCALFrequencyDivisionRatio
[1]-BIT8:
0:Data Streaming Mode, 1:DDS Mode
[1]-BIT9:
0:None, 1:Under DDS Mode, white noise
[1]-BIT10:
0:None, 1:Under DDS Mode, MLS

[2]- DDSPhaseIncremental
[3]- DDSPhaseSweepIncremental
[4]- DDSSweepStartAmplitude
[5]- DDSSweepAmplitudeDC
[6]- ProbCalFrequencyDivisionRatio
For SoundCardDAQ.dll and SoundCardASIODAQO.dll, ReservedDWORD[7] is used to differentiate Device Models:

0: Sound Cards
100: RTX6001
101: RME ADI-2 Pro
102: RME ADI-2/4 PRO SE

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. Under streaming mode, 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. Under DDS mode, it contains one full DDS buffer of data depicting the shape of one cycle of the repetitive output waveform.

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.

For SoundCardASIODAQO.dll:
[2]: Data Format
0: integer
1: 32-bit float format

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. If *dwCallBack* = NULL, then no message will be posted back.

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

2: VT hardware initialization and initialize the DLL with the information stored in the VT hardware
3: download ProbCAL setting when signal generator is not running
4: initialize the DLL with the information stored in the VT hardware, and store DAOOffset in OutputSamplingParameters.ReservedDouble[0], DAOGain in OutputSamplingParameters.ReservedDouble[1], so that the calling procedure can correct the data it outputs.
5: store DAOOffset in OutputSamplingParameters.ReservedDouble[0], DAOGain in OutputSamplingParameters.ReservedDouble[1], so that the calling procedure can correct the data it outputs.

Correction formula(16 bits): NewDAOValue=DAO_DC + (DAOOffset-0x8000) / 0xFFFF + [1 + (DAOGain - 0x8000) / 0xFFFF] × DAO_OSC
7: Open the device's own control panel (e.g. ASIO control panel) if any

Return Values

<0: fail.

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 retrieves 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.

```
int Unlock(
WORD nSerialNumberPart1, //serial number part 1
WORD nSerialNumberPart2, //serial number part 2
WORD nSerialNumberPart3, //serial number part 3
WORD 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.

Return Values

Reserved.

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.

5.4 C++ Wrappers with simple data structures for Labview and others

Some arguments in vtDAO APIs have a complex data structure which may not be readily supported by other software development tools such as Labview. vtDAOLV.dll is a wrapper developed around all those vtDAO compatible dlls. It uses simple data structures.

In Labview, vtDAOLV.dll can be called using Call Library function node. Stdcall (WINAPI) calling convention should be used.

For implicit linking in other software tools, the corresponding header file is vtDAOcplus.h and the library file is vtDAOLV.lib.

5.4.1 DAOLV_SetDAOType

The DAOLV_SetDAOType function loads a hardware-specific vtDAO compatible dll dynamically. This function must be called before any interface functions can be used including the Unlock function.

```
int SetDAOType(
WORD Type
)
```

Parameters

Type

Type of hardware.

0: SoundCardMMEDAO.dll
1: SoundCardASIODAQO.dll
2: NIDAO.dll
3: VTDAO1.dll
4: Reserved
5: MyDAO.dll

Return Values

0: Successful.
1: Failed

5.4.2 DAOLV_Unlock

Same as Section 5.2.7.

5.4.3 DAOLV_SetOutputSamplingParameters

Basically it sets the OutputSamplingParameters in DAO_SetParameters API (Refer to Sections 5.2.1 & 5.1.1).

```
int DAOLV_SetOutputSamplingParameters(
double SamplingFrequency,
WORD SamplingChannels,
WORD SamplingBitResolution,
DWORD BufferLength,
WORD DeviceNo,
WORD * pChannelNo,
double * pHighLimit,
double * pLowLimit,
int Mode,
double Duration,
double * pReservedDouble,
DWORD * pReservedDWORD
)
```


Parameters

Refer to Sections 5.1.1.

Return Values

0: Successful.

5.4.4 DAOLV_SetDAOData

Basically it sets the DAOData in DAO_SetParameters API (Refer to Sections 5.2.1 & 5.1.2).

```
int DAOLV_SetDAOData(
char *pData,
(* DataNotify)(BOOL PrepareDataFlag, BOOL NotifyFlag);
WORD *Status,
double * pReservedDouble,
DWORD * pReservedDWORD,
DWORD dwCallBack,
DWORD fdwOpen
)
```

Parameters

Refer to Section 5.1.2. There are some important differences to note here.

DataNotify(BOOL PrepareDataFlag, BOOL NotifyFlag)

In addition to the description in Section 5.1.2, the callback function address here can be NULL, in which case the message WM_MYMESSAGE_DAO_DATA described in Section 5.3.2 should be used instead under streaming mode.

dwCallBack

Refer to Section 5.2.1.

FdwOpen

Refer to Section 5.2.1. In addition:

0: *dwCallBack* is a window handle, a message will be posted to it whenever a new frame of data is required and / or a frame of data has been output completely, together with the *PrepareDataFlag* indicated by WPARAM and *NotifyFlag* indicated by LPARAM.

1: *dwCallBack* is a thread identifier, a message will be posted to it whenever a new frame of data is required and / or a frame of data has been output completely, together with the *PrepareDataFlag* indicated by WPARAM and *NotifyFlag* indicated by LPARAM.

6: *dwCallBack* is a user event refnum in Labview, a message will be posted to it whenever a new frame of data is required and / or a frame of data has

been output completely, together with the *PrepareDataFlag* indicated by WPARAM.

Return Values

0: Successful.

5.4.5 DAOLV_SetParameters

This function is used to perform some special functions in DAO_SetParameters API when fdwOpen >= 2. DAOLV_SetOutputSamplingParameters, DAOLV_SetDAOData must be called first.

```
int DAOLV_SetParameters(
DWORD fdwOpen
)
```

Parameters

fdwOpen

fdwOpen >= 2. Refer to Section 5.2.1.

Return Values

0: Successful.

5.4.6 DAOLV_Start

Same as Section 5.2.2, except that it will perform hardware-specific initialization additionally first using DAO_SetParameters( ) with fdwOpen = 2, if this initialization has not been done using DAOLV_SetParameters( ) with fdwOpen = 2 yet.

5.4.7 DAOLV_Stop

Same as Section 5.2.3.

5.4.8 DAOLV_AddData

```
int DAOLV_AddData()
```

There are two methods for the calling program to be notified of data preparation requests: one is to provide the address of a callback function `DataNotify` and let it to be called by the called program. However, this method may not be feasible in some cases whereby mixed programming languages are used and it is difficult to pass the address of the callback function from the calling program to the called program. The other is to use the `DAOLV_AddData` function to prepare data upon receiving the message WM_MYMESSAGE_DAO_DATA from the called program.

When the latter method is used, the address of the callback function `DataNotify` should be set to NULL. Under streaming mode, `DAOLV_AddData` should be called three times initially before `DAOLV_Start` is executed if the duration of signal is longer than 3 seconds. This is to provide triple buffering to avoid any discontinuity in the output signal. After that, it should be called once upon receiving a data preparation request via WM_MYMESSAGE_DAO_DATA message. The data pointed by *`pData`* should be prepared first before calling `DAOLV_AddData`. Under DDS mode, it should be called only once before `DAOLV_Start` is executed.

Return Values

0: Successful.

5.4.9 DAOLV_AddDataLV

```
int DAOLV_AddDataLV(char *pData)
```

This function is similar to `DAOLV_AddData()` except that it provides a pointer of the prepared data instead of using the data pointer `pData` passed to the called program via `DAOLV_SetDAOData()`.

5.5 C Wrappers with simple data structures for LabWindows/CVI and others

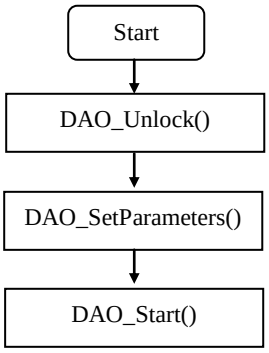
Some software development tools supports ANSI C interfaces only, such as LabWindow/CVI. VtDAOLV.dll provides a set of C interfaces with exactly the same functions and arguments as those described in Section 5.4. They are:

int DAOCVI_SetDAOType
int DAOCVI_Unlock
int DAOCVI_SetOutputSamplingParameters
int DAOCVI_SetDAOData
int DAOCVI_SetParameters
int DAOCVI_Start
int DAOCVI_Stop
int DAOCVI_AddData
int DAOCVI_AddDataLV

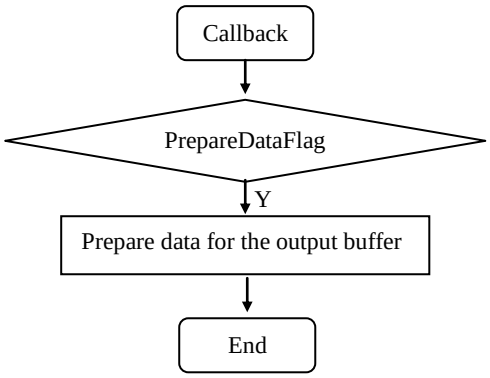
The corresponding header file is vtDAOc.h and the library file is vtDAOLV.lib.

6. vtDAO Development Guide

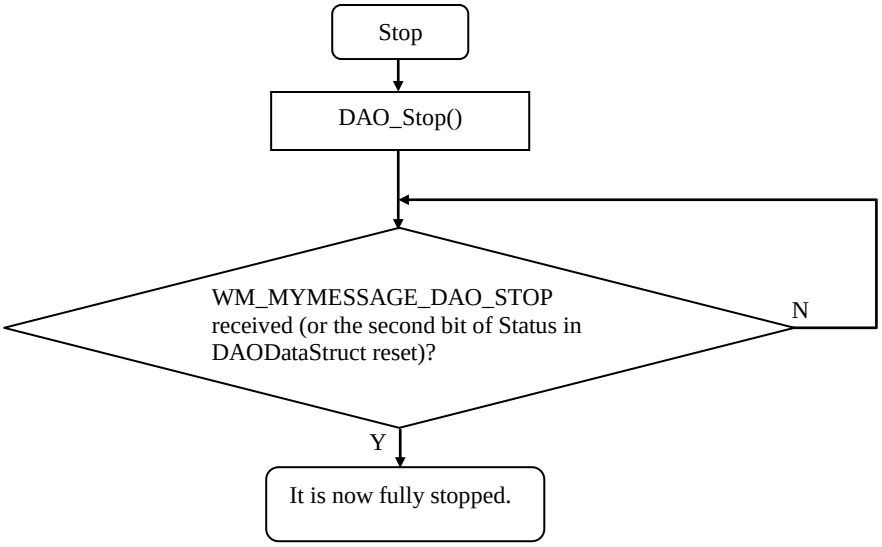
6.1 Flowcharts



Start DAO



CallBack Function DataNotify



Stop DAO

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) VTDAO1.dll for VTDAO1 devices
(5) Any other vtDAO compatible DLLs.

3. vtDAO interface LIBs:
Every dll comes with its own lib file.

6.3 Auxiliary Files

1. vtDAO interface DLLs' C++ Wrappers with simple data structures for Labview and others
   (1) Header file to be included: vtDAOcplus.h
   (2) LIB file: vtDAOLV.lib
   (3) DLL file: vtDAOLV.dll (this is in addition to the basic vtDAO interface DLLs)

2. vtDAO interface DLLs' C Wrappers with simple data structures for LabWindows/CVI and others
   (1) Header file to be included: vtDAOc.h
   (2) LIB file: vtDAOLV.lib
   (3) DLL file: vtDAOLV.dll (this is in addition to the basic vtDAO interface DLLs)

6.4 Hardware Specific Configuration Files

Some hardware devices such as VT DSOs come with some hardware specific configuration files. These files should also be included in the developed software package. For VT DSOs, these files can be found in Multi-Instrument's installation directory\HardwareConfig\….

6.5 How to Choose Correct Output Mode Under Streaming Mode

Under DDS mode, the calling program needs only to prepare data to fill the DDS buffer once before calling `DAO_Start`.

Under streaming mode, the calling program has a few output modes to choose in order not to prepare the same data (if any) repeatedly to save CPU times.

6.5.1 Hardware Sampling Clock

If the data to be output repeats every second (e.g. a signal with an integer frequency), 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 (e.g. a signal with a non-integer frequency), 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`.

6.5.2 Software Timed Sampling Clock

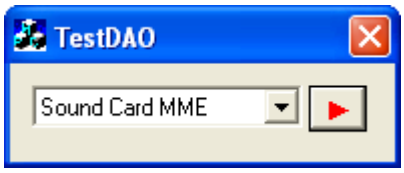
If the data to be output repeats every second, then mode 11 or 12 should be used. If the data output to be stopped automatically by the interface DLL at the end of the specified duration, then mode 11 should be used. If mode 12 is used, the data output will not stop until the calling program calls `DAO_Stop`.

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. vtDAO Sample Programs

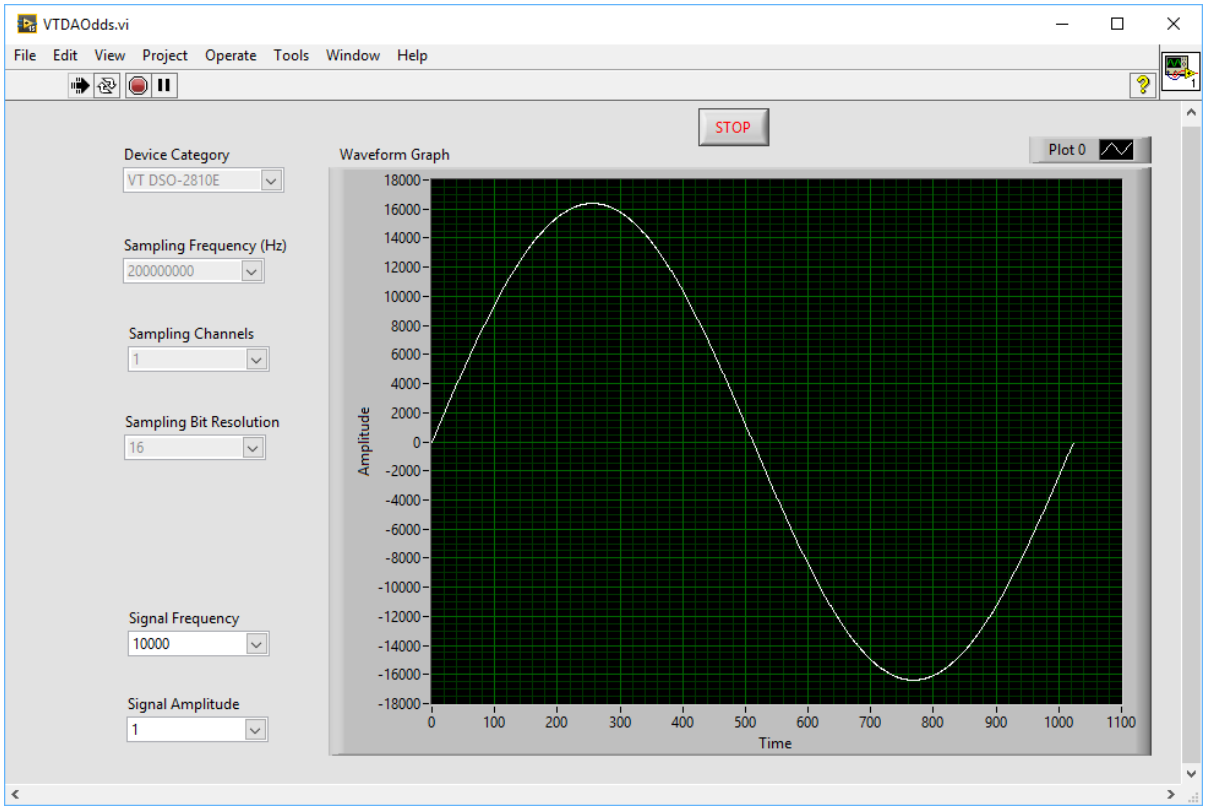
7.1 TestDAO written in Visual C++ 6.0



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 (Graphical User Interface) are provided for changing these parameters for simplicity purpose. Both DDS and streaming modes are demonstrated in this program.

7.2 vtDAO Labview Samples

7.2.1 vtDAOdds.vi written in Labview 15.0



vtDAOdds.vi is a sample VI for Labview programming using vtDAO interfaces. VtDAO interface wrapper: vtDAOLV.dll, is called using Labview Call Library function node. It is basically a signal generator VI with adjustable sampling parameters, signal frequency and signal amplitude. To keep the VI simple, these parameters cannot be changed on-the-fly.

DDS mode is used in this VI. The address of the callback function `DataNotify` is set to NULL and `DAOLV_AddData` is used to prepare the data for the DDS buffer. `DAOLV_AddData` is called only once.

7.2.2 vtDAOstreaming.vi written in Labview 15.0

vtDAOstreaming.vi is a sample VI similar to vtDAOdds.vi introduced previously. However, it uses streaming mode instead. The address of the callback function `DataNotify` is set to NULL and `DAOLV_AddDataLV` is used to prepare the output data. Three buffers of data are prepared before calling `DAOLV_Start`, after which one buffer of data is prepared upon each data preparation message.

7.2.3 vtDAOstreamingSoundCard.vi written in Labview 15.0

This VI is basically the same as vtDAOstreaming.vi. However, its sampling parameters are set, by default, to values compatible with sound cards. It is easier for those who want to try vtDAO Labview development without a VT hardware device.

7.3 TestDAO written in Visual C# 2012

TestDAO_CSharp is a sample DAO back-end program written in Visual C#, with functions similar to its Visual C++ counterpart introduced previously. Instead of calling vtDAO interface dll directly, it interfaces to vtDAOLV.dll which in turn calls the respective vtDAO interface dll, thus avoiding using complex data structures in the original vtDAO APIs. Unlike its VC++ counterpart which uses the callback function `DataNotify` to prepare data, the C# program uses `DAOLV_AddData` instead. Both DDS and streaming modes are demonstrated in this program.

