XZ100

ACQIRIS PROGRAMMER'S REFERENCE MANUAL

December 2004

The information in this document is subject to change without notice and may not be construed as in any way as a commitment by Acqiris. While Acqiris makes every effort to ensure the accuracy and contents of the document it assumes no responsibility for any errors that may appear.

All software described in the document is furnished under license. The software may only be used and copied in accordance with the terms of license. Instrumentation firmware is thoroughly tested and thought to be functional but it is supplied "as is" with no warranty for specified performance. No responsibility is assumed for the use or the reliability of software, firmware or any equipment that is not supplied by Acqiris SA or its affiliated companies.

Any versions of this manual which are supplied with a purchased product will be replaced at your request with the latest revision in electronic format. At Acqiris we appreciate and encourage customer input. If you have a suggestion related to the content of this manual or the presentation of information, please contact your local Acqiris representative or Acqiris Technical Support (support@acqiris.com) or come visit our web site at http://www.acqiris.com.

Trademarks: product and company names listed are trademarks or trade names of their respective companies

Acqiris Headquarters:

Acqiris SA
18, chemin des Aulx
CH-1228 Plan-les-Ouates
Geneva
Switzerland

Tel: +41 22 884 33 90

Fax: +41 22 884 33 99

Acqiris USA:

Acqiris LLC
234 Cromwell Hill Rd.
P.O. Box 2203
Monroe, NY 10950-1430
USA

Tel: 845 782 6544

Fax: 845 782 4745

Acqiris Asia-Pacific:

Acqiris Pty Ltd
Suite 7, Level 1
407 Canterbury Road,
P.O. Box 13
Surrey Hills 3127
Australia

Tel: +61 3 9888 4586

Fax: +61 3 9849 0861

CONTENTS

1. Introduction

1.1. Message to the User

Congratulations on having purchased an Acqiris data conversion product. Acqiris Digitizers, Averagers, and Analyzers are high-speed data acquisition modules designed for capturing high frequency electronic signals. To get the most out of the products we recommend that you read the accompanying product User Manual, the Programmer's Guide and this Programmer's Reference Manual carefully. We trust that the product you have purchased as well as the accompanying software will meet with your expectations and provide you with a high quality solution to your data conversion applications.

1.2. Using this Manual

This guide assumes you are familiar with the operation of a personal computer (PC) running a Windows 95/98/2000/NT4/XP or other supported operating system. In addition you ought to be familiar with the fundamentals of the programming environment that you will be using to control your Acqiris product. It also assumes you have a basic understanding of the principles of data acquisition using either a waveform digitizer or a digital oscilloscope.

The **User Manual** that you also have received (or have access to) has important and detailed instructions concerning your Acqiris product. You should consult it first. You will find the following chapters there:

Chapter 1 ***OUT OF THE BOX,*** describes what to do when you first receive your new Acqiris product. Special attention should be paid to sections on safety, packaging and product handling. Before installing your product please ensure that your system configuration matches or exceeds the requirements specified.

Chapter 2 ***INSTALLATION***, covers all elements of installation and performance verification. Before attempting to use your Acqiris product for actual measurements we strongly recommend that you read all sections of this chapter.

Chapter 3 ***PRODUCT DESCRIPTION***, provides a full description of all the functional elements of your product.

Chapter 4 ***RUNNING THE ACQIRIS DEMONSTRATION APPLICATION***, describes either the operation of AcqirisLive 2.13, an application that enables basic operation of Acqiris digitizers or averagers in a Windows 95/98/2000/NT4/XP environment; the operation of AP_SSRDemo and in the following chapter APx01Demo, applications that enable basic operation of Acqiris analyzers in a Windows 95/98/2000/NT4/XP environment;

The **Programmer's Guide** is divided into 4 separate sections.

Chapter 1 ***INTRODUCTION,*** describes what can be found where in the documentation and how to use it.

Chapter 2 ***PROGRAMMING ENVIRONMENTS & GETTING STARTED***, provides a description for programming applications using a variety of software products and development environments.

Chapter 3 ***PROGRAMMING AN ACQIRIS DIGITIZER***, provides information on using the device driver functions to operate an Acqiris digitizer.

Chapter 4 ***ATTRIBUTES***, contains reference information about attributes. The attribute interface to the driver can be used with the MATLAB interface and the SP201 Software Development Kit.

This Programmer's Reference manual is divided into 2 sections.

Chapter 1 ***INTRODUCTION,*** describes what can be found where in the documentation and how to use it.

Chapter 2 ***DEVICE DRIVER FUNCTION REFERENCE***, contains a full device driver function reference. This documents the traditional Application Program Interface (API) as it can be used in the following environments:

LabWindowsCVI, Visual C++, LabVIEW, Visual Basic, Visual Basic .NET.

1.3. Conventions Used in This Manual

The following conventions are used in this manual:

⚠ This icon to the left of text warns that an important point must be observed.

WARNING Denotes a warning, which advises you of precautions to take to avoid being electrically shocked.

CAUTION Denotes a caution, which advises you of precautions to take to avoid electrical, mechanical, or operational damages.

NOTE Denotes a note, which alerts you to important information.

Italic text denotes a warning, caution, or note.

Bold Italic text is used to emphasize an important point in the text or a note

`mono` text is used for sections of code, programming examples and operating system commands.

Certain features are common to several different modules. For increased readability we have defined the following families:

DC271-FAMILY	DC135/DC140/DC211/DC211A/DC241/DC241A/ DC271/DC271A/DP214/DP235/DP240
AP-FAMILY	AP240/AP235/AP100/AP101/AP200/AP201
12-bit-FAMILY	DC440/DC438/DC436/DP310/DP308/DP306

1.4. Warning Regarding Medical Use

The Digitizer cards are not designed with components and testing procedures that would ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of these cards involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user. These cards are ***not*** intended to be a substitute for any form of established process or equipment used to monitor or safeguard human health and safety in medical treatment.

⚠ ***WARNING:*** ***The modules discussed in this manual have not been designed for making direct measurements on the human body. Users who connect an Acqiris module to a human body do so at their own risk.***

1.5. Warranty

Please refer to the appropriate User Manual.

1.6. Warranty and Repair Return Procedure, Assistance and Support

Please refer to the appropriate User Manual.

1.7. System Requirements

Please refer to the appropriate User Manual.

2. Device Driver Function Reference

All function calls require the argument **instrumentID** in order to identify the Acqiris Digitizer card to which the call is directed. The only exceptions are the initialization/termination functions:

- **AcqrsD1_closeAll**
- **AcqrsD1_getNbrPhysicalInstruments**
- **AcqrsD1_setSimulationOptions**
- **AcqrsD1_init**
- **AcqrsD1_multiInstrAutoDefine**
- **AcqrsD1_multiInstrUndefineAll**
- **AcqrsD1_InitWithOptions**

The functions **AcqrsD1_init**, **AcqrsD1_InitWithOptions** and **AcqrsD1_multiInstrDefine** actually return instrument identifiers at initialization time, for subsequent use in the other function calls.

2.1. Status values and Error codes

All function calls return a status value of type '**ViStatus**' with information about the success or failure of the call. All Acqiris specific values can be found in the header file **AcqirisErrorCodes.h** and are shown in Table 2-1. The generic ones, defined by the VXIplug&play Systems Alliance, are listed in the header file **vpptype.h** (VXIplug&play instrument driver header file, which includes **visatype.h**: fundamental VISA data types and macro definitions). They are reproduced in Table 2-2 for convenience. The header file **AcqirisD1Interface.h** shows the common error codes associated with each function.

Acqiris Error Codes	Hex value	Decimal value
ACQIRIS_ERROR_FILE_NOT_FOUND	BFFA4800	-1074116608
ACQIRIS_ERROR_PATH_NOT_FOUND	BFFA4801	-1074116607
ACQIRIS_ERROR_INVALID_HANDLE	BFFA4803	-1074116605
ACQIRIS_ERROR_NOT_SUPPORTED	BFFA4805	-1074116603
ACQIRIS_ERROR_INVALID_WINDOWS_PARAM	BFFA4806	-1074116602
ACQIRIS_ERROR_NO_DATA	BFFA4807	-1074116601
ACQIRIS_ERROR_NO_ACCESS	BFFA4808	-1074116600
ACQIRIS_ERROR_BUFFER_OVERFLOW	BFFA4809	-1074116599
ACQIRIS_ERROR_ALREADY_OPEN	BFFA4840	-1074116544
ACQIRIS_ERROR_SETUP_NOT_AVAILABLE	BFFA4880	-1074116480
ACQIRIS_ERROR_IO_WRITE	BFFA48A0	-1074116448
ACQIRIS_ERROR_IO_READ	BFFA48A1	-1074116447
ACQIRIS_ERROR_INTERNAL_DEVICENO_INVALID	BFFA48C0	-1074116416
ACQIRIS_ERROR_TOO_MANY_DEVICES	BFFA48C1	-1074116415
ACQIRIS_ERROR_EEPROM_DATA_INVALID	BFFA48C2	-1074116414
ACQIRIS_ERROR_INIT_STRING_INVALID	BFFA48C3	-1074116413
ACQIRIS_ERROR_INSTRUMENT_NOT_FOUND	BFFA48C4	-1074116412
ACQIRIS_ERROR_INSTRUMENT_RUNNING	BFFA48C5	-1074116411
ACQIRIS_ERROR_INSTRUMENT_STOPPED	BFFA48C6	-1074116410
ACQIRIS_ERROR_MODULES_NOT_ON_SAME_BUS	BFFA48C7	-1074116409
ACQIRIS_ERROR_NOT_ENOUGH_DEVICES	BFFA48C8	-1074116408
ACQIRIS_ERROR_NO_MASTER_DEVICE	BFFA48C9	-1074116407
ACQIRIS_ERROR_PARAM_STRING_INVALID	BFFA48CA	-1074116406
ACQIRIS_ERROR_COULD_NOT_CALIBRATE	BFFA48CB	-1074116405
ACQIRIS_ERROR_CANNOT_READ_THIS_CHANNEL	BFFA48CC	-1074116404
ACQIRIS_ERROR_PRETRIGGER_STILL_RUNNING	BFFA48CD	-1074116403
ACQIRIS_ERROR_CALIBRATION_FAILED	BFFA48CE	-1074116402
ACQIRIS_ERROR_MODULES_NOT_CONTIGUOUS	BFFA48CF	-1074116401
ACQIRIS_ERROR_INSTRUMENT_ACQ_LOCKED	BFFA48D0	-1074116400
ACQIRIS_ERROR_INSTRUMENT_ACQ_NOT_LOCKED	BFFA48D1	-1074116399
ACQIRIS_ERROR_INVALID_GEOMAP_FILE	BFFA48E0	-1074116384
ACQIRIS_ERROR_ACQ_TIMEOUT	BFFA4900	-1074116352
ACQIRIS_ERROR_OVERLOAD	BFFA4901	-1074116351
ACQIRIS_ERROR_PROC_TIMEOUT	BFFA4902	-1074116350
ACQIRIS_ERROR_LOAD_TIMEOUT	BFFA4903	-1074116349
ACQIRIS_ERROR_READ_TIMEOUT	BFFA4904	-1074116348
ACQIRIS_ERROR_FPGA_1_LOAD	BFFA4A01	-1074116095
ACQIRIS_ERROR_FPGA_2_LOAD	BFFA4A02	-1074116094
ACQIRIS_ERROR_FPGA_3_LOAD	BFFA4A03	-1074116093
ACQIRIS_ERROR_FPGA_4_LOAD	BFFA4A04	-1074116092
ACQIRIS_ERROR_FPGA_5_LOAD	BFFA4A05	-1074116091
ACQIRIS_ERROR_FPGA_6_LOAD	BFFA4A06	-1074116090
ACQIRIS_ERROR_FPGA_7_LOAD	BFFA4A07	-1074116089

Acqiris Error Codes	Hex value	Decimal value
ACQIRIS_ERROR_FPGA_8_LOAD	BFFA4A08	-1074116088
ACQIRIS_ERROR_ATTR_NOT_FOUND	BFFA4B00	-1074115840
ACQIRIS_ERROR_ATTR_WRONG_TYPE	BFFA4B01	-1074115839
ACQIRIS_ERROR_ATTR_IS_READ_ONLY	BFFA4B02	-1074115838
ACQIRIS_ERROR_ATTR_IS_WRITE_ONLY	BFFA4B03	-1074115837
ACQIRIS_ERROR_ATTR_ALREADY_DEFINED	BFFA4B04	-1074115836
ACQIRIS_ERROR_ATTR_IS_LOCKED	BFFA4B05	-1074115835
ACQIRIS_ERROR_ATTR_INVALID_VALUE	BFFA4B06	-1074115834
ACQIRIS_ERROR_OTHER_WINDOWS_ERROR	BFFA4C00	-1074115584
ACQIRIS_ERROR_UNKNOWN_ERROR	BFFA4C01	-1074115583
ACQIRIS_ERROR_PARAMETER9	BFFA4D09	-1074115319
ACQIRIS_ERROR_PARAMETER10	BFFA4D0A	-1074115318
ACQIRIS_ERROR_PARAMETER11	BFFA4D0B	-1074115317
ACQIRIS_ERROR_PARAMETER12	BFFA4D0C	-1074115316
ACQIRIS_ERROR_PARAMETER13	BFFA4D0D	-1074115315
ACQIRIS_ERROR_PARAMETER14	BFFA4D0E	-1074115314
ACQIRIS_ERROR_PARAMETER15	BFFA4D0F	-1074115313
ACQIRIS_ERROR_NBR_SEG	BFFA4D10	-1074115312
ACQIRIS_ERROR_NBR_SAMPLE	BFFA4D11	-1074115311
ACQIRIS_ERROR_DATA_ARRAY	BFFA4D12	-1074115310
ACQIRIS_ERROR_SEG_DESC_ARRAY	BFFA4D13	-1074115309
ACQIRIS_ERROR_FIRST_SEG	BFFA4D14	-1074115308
ACQIRIS_ERROR_SEG_OFF	BFFA4D15	-1074115307
ACQIRIS_ERROR_FIRST_SAMPLE	BFFA4D16	-1074115306
ACQIRIS_ERROR_DATATYPE	BFFA4D17	-1074115305
ACQIRIS_ERROR_READMODE	BFFA4D18	-1074115304
ACQIRIS_WARN_SETUP_ADAPTED	3FFA4E00	1073368576
ACQIRIS_WARN_READPARA_NBRSEG_ADAPTED	3FFA4E10	1073368592
ACQIRIS_WARN_READPARA_NBRSAMP_ADAPTED	3FFA4E11	1073368593
ACQIRIS_WARN_EEPROM_AND_DLL_MISMATCH	3FFA4E12	1073368594
ACQIRIS_WARN_ACTUAL_DATASIZE_ADAPTED	3FFA4E13	1073368595
ACQIRIS_WARN_UNEXPECTED_TRIGGER	3FFA4E14	1073368596

Table 2-1 Acqiris Error Codes

Error code	Hex value	Decimal value
VI_SUCCESS	0	0
VI_ERROR_PARAMETER1	BFFC0001	-1074003967
VI_ERROR_PARAMETER2	BFFC0002	-1074003966
VI_ERROR_PARAMETER3	BFFC0003	-1074003965
VI_ERROR_PARAMETER4	BFFC0004	-1074003964
VI_ERROR_PARAMETER5	BFFC0005	-1074003963
VI_ERROR_PARAMETER6	BFFC0006	-1074003962
VI_ERROR_PARAMETER7	BFFC0007	-1074003961
VI_ERROR_PARAMETER8	BFFC0008	-1074003960
VI_ERROR_FAIL_ID_QUERY	BFFC0011	-1074003951
VI_ERROR_INV_RESPONSE	BFFC0012	-1074003950

Table 2-2 VXIplug&play Error Codes

If important parameters supplied by the user (e.g. an **instrumentID**) are found to be invalid, most functions do not execute and return an error code of the type **VI_ERROR_PARAMETER**i, where i = 1, 2,... corresponds to the argument number.

If the user attempts (with a function **AcqrsD1_configXXXX**) to set a digitizer parameter to a value outside of its acceptable range, the function typically adapts the parameter to the closest allowed value and returns **ACQIRIS_WARN_SETUP_ADAPTED**. The digitizer parameters that are actually in use can be retrieved with the query functions **AcqrsD1_getXXXX**.

Data are always returned through pointers to user-allocated variables or arrays.

Some parameters are labeled "Currently ignored". It is recommended to supply the value "0" (`ViInt32`) or "0.0" (`ViReal64`) in order to be compatible with future products that may offer additional functionality.

2.2. API Function classification

Initialization Functions	*Function Name*
Number of Physical Instruments	`AcqrsD1_getNbrPhysicalInstruments`
MultiInstrument Auto Define	`AcqrsD1_multiInstrAutoDefine`
Initialization	`AcqrsD1_init`
Initialization with Options	`AcqrsD1_InitWithOptions`
Simulation Options	`AcqrsD1_setSimulationOptions`
Calibration Functions	
Calibrate Instrument	`AcqrsD1_calibrate`
Calibrate for External Clock	`AcqrsD1_calibrateEx`
Configuration Functions	
Configure Vertical Settings	`AcqrsD1_configVertical`
Configure Horizontal Settings	`AcqrsD1_configHorizontal`
Configure Channel Combination	`AcqrsD1_configChannelCombination`
Configure Trigger Class	`AcqrsD1_configTrigClass`
Configure Trigger Source	`AcqrsD1_configTrigSource`
Configure Trigger TV	`AcqrsD1_configTrigTV`
Configure Memory Settings	`AcqrsD1_configMemory`
Configure External Clock	`AcqrsD1_configExtClock`
Configure Digitizer Mode	`AcqrsD1_configMode`
Configure Multiplexer Input	`AcqrsD1_configMultiInput`
Configure Control IO	`AcqrsD1_configControlIO`
Configure Frequency Counter	`AcqrsD1_configFCounter`
Configure Averager Configuration Attribute	`AcqrsD1_configAvgConfig`
Configure (program) on-board FPGA	`AcqrsD1_configLogicDevice`
Configure Array of Setup Parameters	`AcqrsD1_configSetupArray`
MultiInstrument Manual Define	`AcqrsD1_multiInstrDefine`
MultiInstrument Undefine	`AcqrsD1_multiInstrUndefineAll`
Acquisition Control Functions	
Start Acquisition	`AcqrsD1_acquire`
Start Acquisition (Extended)	`AcqrsD1_acquireEx`
Query Acquisition Status	`AcqrsD1_acqDone`
Software Trigger	`AcqrsD1_forceTrig`
Software Trigger (Extended)	`AcqrsD1_forceTrigEx`
Stop Acquisition	`AcqrsD1_stopAcquisition`
Wait for End of Acquisition	`AcqrsD1_waitForEndOfAcquisition`
Number of Acquired Segments	`AcqrsD1_reportNbrAcquiredSegments`
Data Transfer Functions	
Universal Waveform Read	`AcqrsD1_readData`
Accumulate Data	`AcqrsD1_accumulateData`
Averaged Data	`AcqrsD1_averagedData`

Read Frequency Counter	`AcqrsD1_readFCounter`
DEPRECATED	**`DO NOT USE FOR NEW PROGRAMS`**
Read Sequence (ADC counts)	`AcqrsD1_readCharSequence`
Read Sequence (Volts)	`AcqrsD1_readRealSequence`
Read Waveform (ADC counts)	`AcqrsD1_readCharWform`
Read Waveform (Volts)	`AcqrsD1_readRealWform`
Accumulate Waveform	`AcqrsD1_accumulateWform`
Averaged Waveform	`AcqrsD1_averagedWform`
Query Functions	
Query External Clock	`AcqrsD1_getExtClock`
Query Horizontal Settings	`AcqrsD1_getHorizontal`
Query Channel Combination	`AcqrsD1_getChannelCombination`
Query Memory Settings	`AcqrsD1_getMemory`
Query Multiplexer Input	`AcqrsD1_getMultiInput`
Query Trigger Class	`AcqrsD1_getTrigClass`
Query Trigger Source	`AcqrsD1_getTrigSource`
Query Trigger TV	`AcqrsD1_getTrigTV`
Query Vertical Settings	`AcqrsD1_getVertical`
Query Digitizer Mode	`AcqrsD1_getMode`
Query Control IO	`AcqrsD1_getControlIO`
Query Frequency Counter	`AcqrsD1_getFCounter`
Query Averager Configuration	`AcqrsD1_getAvgConfig`
Instrument Basic Data	`AcqrsD1_getInstrumentData`
Instrument Information	`AcqrsD1_getInstrumentInfo`
Number of Channels	`AcqrsD1_getNbrChannels`
Query Array of Setup Parameters	`AcqrsD1_getSetupArray`
Control Functions	
Query (on-board) Processing Status	`AcqrsD1_procDone`
Start (on-board) Processing	`AcqrsD1_processData`
Stop (on-board) Processing	`AcqrsD1_stopProcessing`
Wait for End of (on-board) Processing	`AcqrsD1_waitForEndOfProcessing`
Utility Functions	
Best Nominal Samples	`AcqrsD1_bestNominalSamples`
Best Sampling Interval	`AcqrsD1_bestSampInterval`
Version	`AcqrsD1_getVersion`
Error Message	`AcqrsD1_errorMessage`
Reset	`AcqrsD1_reset`
Reset Digitizer Memory	`AcqrsD1_resetDigitizerMemory`
Restore Internal Registers	`AcqrsD1_restoreInternalRegisters`
Set LED Color	`AcqrsD1_setLEDColor`
Close all instruments	`AcqrsD1_closeAll`

2.3. API Function descriptions

This section describes each function in the Device Driver. The functions appear in alphabetical order.

2.3.1 AcqrsD1_accumulateData

Purpose

Returns a waveform as an array and accumulates it in a client array.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
readPar	AqReadParameters	Requested parameters for the acquired waveform.

Output

Name	Type	Description
dataArray	ViAddr	User-allocated waveform destination array of type char or byte. Its size in dataType units MUST be at least 'nbrSamples' + 32, for reasons of data alignment.
sumArray	ViInt32 [ ]	User-allocated waveform accumulation array. Its size MUST be at least 'nbrSamples'. It is a 32-bit integer (long) array, with the sample-by-sample sum of the data values in ADC count unit (LSB). See discussion below.
dataDesc	AqDataDescriptor	Waveform descriptor structure.
segDescArray	ViAddr	Segment descriptor structure.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function uses the AcqrsD1_readData routine. However, only 'readPar->nbrSegments = 1' and 'readPar->readMode = 0' (ReadModeStdW) are supported. 'readPar->dataType = 3' (real) and 'readPar->dataType = 2' (long) are NOT supported.

The **sumArray** contains the sample-by-sample sums. To get the average values, the array elements must be divided by the number of accumulations performed. The sumArray can be interpreted as an unsigned integer. Alternatively, negative values have to be increased by 2**32.

The number of acquisitions, nbrAcq, can be at most 16777216 for 'readPar->dataType = 0' (char) or 65536 for 'readPar->dataType = 1' (short). This is to avoid an overflow where the summed values will wrap around 0.

The value in Volts of a data point **data** in the returned **dataArray** can be computed with the formula:

```
V = dataDesc.vGain * data – dataDesc.vOffset
```

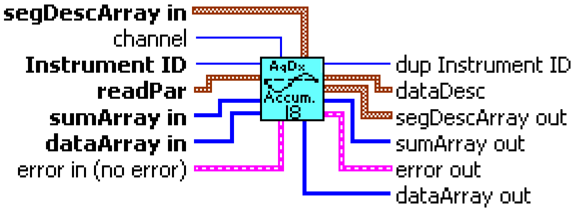
LabWindowsCVI/Visual C++ Representation

```
ViStatus AcqrsD1_accumulateData (ViSession instrumentID,
                     ViInt32 channel, AqReadParameters* readPar,
                     void* dataArray, ViInt32 sumArray[],
                     AqDataDescriptor* dataDesc,
                     void* segDescArray);
```

LabVIEW Representation

```
AqDx Accumulate Data.vi
```

This Vi is polymorphic, the sample data is returned in an array of type I8 or I16.



Visual Basic Representation

```
AccumulateData  (ByVal instrumentID As Long, _
                 ByVal channel As Long, _
                 readPar As AqReadParameters, _
                 dataArray As Any, _
                 sumArray As Long, _
                 dataDesc As AqDataDescriptor, _
                 segDescArray As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_accumulateData (ByVal instrumentID As Int32, _
                      ByVal channel As Int32, _
                      ByRef readPar As AqReadParameters, _
                      ByRef dataArray As Byte, _
                      ByRef sumArray As Int32, _
                      ByRef dataDesc As AqDataDescriptor, _
                      ByRef segDescArray As AqSegmentDescriptor) _
               As  Int32
```

2.3.2 AcqrsD1_accumulateWform (DEPRECATED)

Purpose

Returns a waveform as a byte (8-bit integer) array and accumulates it in a client array. This routine is for use with 8-bit Digitizers.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
segmentNumber	ViInt32	Requested segment number, may assume 0 to the (number of segments – 1) set with the function **`AcqrsD1_configMemory`**.
firstSample	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1) set with the function **`AcqrsD1_configMemory`**.
nbrSamples	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **`AcqrsD1_configMemory`**.

Output

Name	Type	Description
waveformArray	ViChar [ ]	User-allocated waveform destination array of type char or byte. Its size MUST be at least 'nbrSamples' + 32, for reasons of data alignment.
sumArray	ViInt32 [ ]	User-allocated waveform accumulation array. Its size MUST be at least 'nbrSamples'. It is a 32-bit integer (long) array, with the sample-by-sample sum of the data values in ADC count unit (LSB). See discussion below.
returnedSamples	ViInt32	Number of data samples actually returned
sampTime	ViReal64	Sampling interval in seconds
vGain	ViReal64	Vertical gain in Volts/LSB. See discussion below.
vOffset	ViReal64	Vertical offset in Volts. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The **sumArray** contains the sample-by-sample sums. To get the average values, the array elements must be divided by the number of accumulations performed.

The value in Volts of a data point **data** in the returned **waveformArray** can be computed with the formula:

```
V = vGain * data - vOffset
```

LabWindowsCVI/Visual C++ Representation

```
ViStatus AcqrsD1_accumulateWform (ViSession instrumentID,
                     ViInt32 channel, ViInt32 segmentNumber,
                     ViInt32 firstSample, ViInt32 nbrSamples,
                     ViChar waveformArray[], ViInt32 sumArray[],
                     ViInt32 *returnedSamples, ViReal64 *sampTime,
                     ViReal64 *vGain, ViReal64 *vOffset);
```

LabVIEW Representation

```
AqDx Read Accumulated Waveform.vi should be considered as obsolete.
Please use AqDx Accumulate Data.vi instead.
```

Visual Basic Representation

```
AccumulateWform (ByVal instrumentID As Long, _
                 ByVal channel As Long, _
                 ByVal segmentNumber As Long, _
                 ByVal firstSample As Long, _
                 ByVal nbrSamples As Long, _
                 waveformArray As Byte, _
                 sumArray As Long, _
                 returnedSamples As Long, _
                 sampTime As Double, _
                 vGain As Double, _
                 vOffset As Double) As Long
```

2.3.3 AcqrsD1_acqDone

Purpose

Checks if the acquisition has terminated.

Parameters

Input

Name	Type	Description
InstrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
Done	ViBoolean	done = VI_TRUE if the acquisition is terminated VI_FALSE otherwise

Return Value

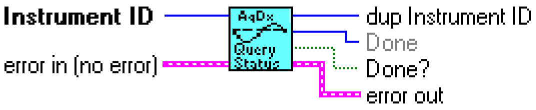
Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_acqDone(ViSession instrumentID,
                                  ViBoolean* done);
```

LabVIEW Representation

```
AqDx Query Acquisition Status.vi
```



Visual Basic Representation

```
AcqDone (ByVal instrumentID As Long, done As Boolean) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_acqDone (ByVal instrumentID As Int32, _
                              ByRef done As Boolean) As Int32
```

2.3.4 AcqrsD1_acquire

Purpose

Starts an acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

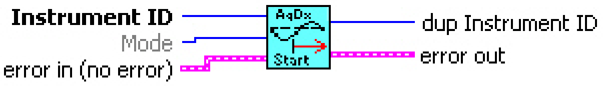
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_acquire(ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Start Acquisition.vi
```



Visual Basic Representation

```
Acquire (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_acquire (ByVal instrumentID As Int32) As Int32
```

2.3.5 AcqrsD1_acquireEx

Purpose

Starts an acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
acquireMode	ViInt32	= 0, normal = 2, continue to accumulate (AP Averagers only)
acquireFlags	ViInt32	Flags, currently not used
acquireParams	ViInt32	Parameters, currently not used
reserved	ViInt32	Currently not used

Return Value

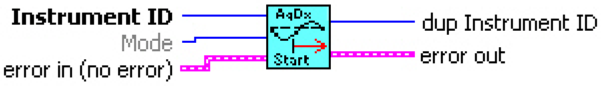
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_acquireEx(ViSession instrumentID ,
                     ViInt32 acquireMode, ViInt32 acquireFlags,
                     ViInt32 acquireParams, ViInt32 reserved);
```

LabVIEW Representation

```
AqDx Start Acquisition.vi
```



Visual Basic Representation

```
AcquireEx (ByVal instrumentID As Long, ByVal acquireMode As Long, _
          ByVal acquireFlags As Long, ByVal acquireParams As Long, _
          ByVal reserved As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_acquireEx (ByVal instrumentID As Int32, _
    ByVal acquireMode As Int32, ByVal acquireFlags As Int32, _
    ByVal acquireParams As Int32, ByVal reserved As Int32) As Int32
```

2.3.6 AcqrsD1_averagedData

Purpose

This function is intended for single instrument, single channel operation.

Perform a series of acquisitions and get the resulting averaged waveform.

Parameters

Input

Name	Type	Description
InstrumentID	ViSession	Instrument identifier
Channel	ViInt32	1...Nchan
readPar	AqReadParameters	Requested parameters for the acquired waveform
nbrAcq	ViInt32	Number of acquisitions to be performed.
calculateMean	ViBoolean	TRUE to divide the sumArray by nbrAcq to get the mean values. FALSE to leave the sample-by-sample sums in the sumArray.
timeout	ViReal64	Acquisition timeout in seconds. The function will return an error if, for each acquisition, no trigger arrives within the specified timeout after the start of the acquisition. The minimum value is 1 ms.

Output

Name	Type	Description
dataArray	ViAddr	User-allocated waveform destination array of type char or byte. Its size in dataType units MUST be at least 'nbrSamples' + 32, for reasons of data alignment.
sumArray	ViInt32 [ ]	User-allocated waveform accumulation array. Its size MUST be at least 'nbrSamples'. It is a 32-bit integer (long) array, with the sample-by-sample sum of the data values in ADC count unit (LSB). See discussion below.
dataDesc	AqDataDescriptor	Waveform descriptor structure. The returned values will be those of the last acquisition
segDescArray	ViAddr	Segment descriptor structure. The returned values will be those of the last acquisition.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

Because the acquisition control loop is done inside this function, it is suitable *only* for single instrument, single channel operation.

This function uses the AcqrsD1_readData routine. However, only 'readPar->nbrSegments = 1' and 'readPar->readMode = 0' (ReadModeStdW) are supported. 'readPar->dataType = 3' (real) and 'readPar->dataType = 2' (long) are NOT supported.

The **sumArray** contains either the average values (calculateMean = TRUE), or the sample-by-sample sums (calculateMean = FALSE). Note that, in the latter case, the sumArray can be interpreted as an unsigned integer. Alternatively, negative values have to be increased by 2**32.

The number of acquisitions, nbrAcq, can be at most 16777216 for 'readPar->dataType = 0' (char) or 65536 for 'readPar->dataType = 1' (short). This is to avoid an overflow where the summed values will wrap around 0.

The value in Volts of a data point **data** in the returned **waveformArray** or normalized **sumArray** can be computed with the formula:

```
V = dataDesc.vGain * data – dataDesc.vOffset
```

The function will return ACQIRIS_ERROR_ACQ_TIMEOUT if there is no trigger within the specified timeout interval after the start of each acquisition.

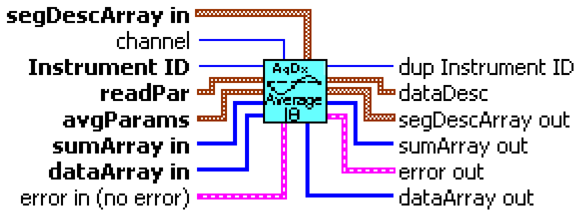
LabWindowsCVI/Visual C++ Representation

```
ViStatus AcqrsD1_averagedData(ViSession instrumentID,
                    ViInt32 channel, AqReadParameters* readPar,
                    ViInt32 nbrAcq, ViInt8 calculateMean,
                    ViReal64 timeout,
                    void* dataArray, ViInt32 sumArray[],
                    AqDataDescriptor* dataDesc,
                    void* segDescArray);
```

LabVIEW Representation

AqDx Averaged Data.vi

This Vi is polymorphic, the sample data is returned in an array of type I8 or I16.



Visual Basic Representation

```
AveragedData  (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               readPar As AqReadParameters, _
               ByVal nbrAcq As Long, _
               ByVal calculateMean As Boolean, _
               ByVal timeout As Double, _
               dataArray As Any, _
               sumArray As Long, _
               dataDesc As AqDataDescriptor, _
               segDescArray As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_averagedData  (ByVal instrumentID As Int32, _
               ByVal channel As Int32, _
               ByRef readPar As AqReadParameters, _
               ByVal nbrAcq As Int32, _
               ByVal calculateMean As Boolean, _
               ByVal timeout As Double, _
               ByRef dataArray As Byte, _
               ByRef sumArray As Int32, _
               ByRef dataDesc As AqDataDescriptor, _
               ByRef segDescArray As AqSegmentDescriptor) As Int32
```

2.3.7 AcqrsD1_averagedWform (DEPRECATED)

Purpose

This function is intended for single instrument, single channel operation. It is for use with 8-bit Digitizers.

Perform a series of acquisitions and get the resulting averaged waveform.

Parameters

Input

Name	Type	Description
InstrumentID	ViSession	Instrument identifier
Channel	ViInt32	1...Nchan
SegmentNumber	ViInt32	Requested segment number, may assume 0 to the (number of segments – 1) set with the function **`AcqrsD1_configMemory`**.
firstSample	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1) set with the function **`AcqrsD1_configMemory`**.
nbrSamples	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **`AcqrsD1_configMemory`**.
nbrAcq	ViInt32	Number of acquisitions to be performed.
timeout	ViReal64	Acquisition timeout in seconds. The function will return an error if, for each acquisition, no trigger arrives within the specified timeout after the start of the acquisition. The minimum value is 1 ms.

Output

Name	Type	Description
waveformArray	ViChar [ ]	User-allocated waveform destination array of type char or byte. Its size MUST be at least 'nbrSamples' + 32, for reasons of data alignment.
sumArray	ViInt32 [ ]	User-allocated waveform accumulation array. Its size MUST be at least 'nbrSamples'. It is a 32-bit integer (long) array, with the sample-by-sample sum of the data values in ADC count unit (LSB). See discussion below.
returnedSamples	ViInt32	Number of data samples actually returned
sampTime	ViReal64	Sampling interval in seconds
vGain	ViReal64	Vertical gain in Volts/LSB. See discussion below.
vOffset	ViReal64	Vertical offset in Volts. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

Because the acquisition control loop is done inside this function, it is suitable *only* for single instrument, single channel operation.

The **sumArray** contains the sample-by-sample sums. To get the average values, the array elements must be divided by **nbrAcq**.

The value in Volts of a data point **data** in the returned **waveformArray** can be computed with the formula:

```
V = vGain * data - vOffset
```

LabWindowsCVI/Visual C++ Representation

```
ViStatus AcqrsD1_averagedWform (ViSession instrumentID,
                     ViInt32 channel, ViInt32 segmentNumber,
                     ViInt32 firstSample, ViInt32 nbrSamples,
                     ViInt32 nbrAcq, ViReal64 timeout,
                     ViChar waveformArray[], ViInt32 sumArray[],
                     ViInt32 *returnedSamples, ViReal64 *sampTime,
                     ViReal64 *vGain, ViReal64 *vOffset);
```

LabVIEW Representation

```
AqDx Read Averaged Waveform.vi should be considered to be obsolete.
Please use AqDx Averaged Data.vi instead.
```

Visual Basic Representation

```
AveragedWform (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               ByVal segmentNumber As Long, _
               ByVal firstSample As Long, _
               ByVal nbrSamples As Long, _
               ByVal nbrAcq As Long, _
               ByVal timeout As Double, _
               waveformArray As Byte, _
               sumArray As Long, _
               returnedSamples As Long, _
               sampTime As Double, _
               vGain As Double, _
               vOffset As Double) As Long
```

2.3.8 AcqrsD1_bestNominalSamples

Purpose

Helper function to simplify digitizer configuration. It returns the maximum nominal number of samples that fit into the available memory.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nomSamples	ViInt32	Maximum number of data samples available

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

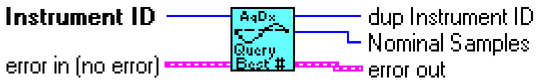
When using this method, make sure to use **AcqrsD1_configHorizontal** and **AcqrsD1_configMemory** beforehand to set the sampling rate and the number of segments to the desired values (**nbrSamples** in **AcqrsD1_configMemory** may be any number!). **AcqrsD1_bestNominalSamples** depends on these variables.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_bestNominalSamples(ViSession instrumentID,
                        ViInt32* nomSamples);
```

LabVIEW Representation

```
AqDx Query Best Nominal Samples.vi
```



Visual Basic Representation

```
BestNominalSamples (ByVal instrumentID As Long, _
                      nomSamples As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_bestNominalSamples (ByVal instrumentID As Int32, _
                      ByRef nomSamples As Int32) As Int32
```

2.3.9 AcqrsD1_bestSampInterval

Purpose

Helper function to simplify digitizer configuration. It returns the best possible sampling rate for an acquisition, which covers the **timeWindow** with no more than **maxSamples**. The calculation takes into account the current state of the instrument, in particular the requested number of segments. In addition, this routine returns the "real" nominal number of samples that can be accommodated (it is computed as **timeWindow**/**samplingInterval**!).

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
maxSamples	ViInt32	Maximum number of samples to be used
timeWindow	ViReal64	Time window to be covered, in seconds

Output

Name	Type	Description
sampInterval	ViReal64	Recommended sampling interval in seconds
nomSamples	ViInt32	Recommended number of data samples

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The function returns the value status = ACQIRIS_ERROR_SETUP_NOT_AVAILABLE when the available memory is too short, and the longest available sampling interval too short. The returned sampling interval is the longest one possible. It returns VI_SUCCESS when a good solution has been found.

NOTE: This function *does not* modify the state of the digitizer at all. It simply returns a recommendation that the user is free to override.

NOTE: When using this method, make sure to use **AcqrsD1_configMemory** beforehand to set the number of segments to the desired value (**nbrSamples** may be any number!). **AcqrsD1_bestSampInterval** depends on this variable.

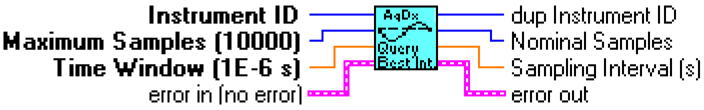
NOTE: The returned "recommended" values for the sampling interval **sampInterval** and the nominal number of samples **nomSamples** are expected to be used for configuring the instrument with calls to **AcqrsD1_configMemory** and **AcqrsD1_configHorizontal**. Make sure to use the same number of segments in this second call to **AcqrsD1_configMemory,** as in the first one.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_bestSampInterval(ViSession instrumentID,
                        ViInt32 maxSamples, ViReal64 timeWindow,
                        ViReal64* sampInterval, ViInt32* nomSamples);
```

LabVIEW Representation

```
AqDx Query Best Sampling Interval.vi
```



Visual Basic Representation

```
BestSampInterval (ByVal instrumentID As Long, _
                  ByVal maxSamples As Long, _
                  ByVal timeWindow As Double, _
                  sampInterval As Double, _
                  nomSamples As Long) As Long
```

Visual Representation

```
AcqrsD1_bestSampInterval (ByVal instrumentID As Int32, _
                  ByVal maxSamples As Int32, _
                  ByVal timeWindow As Double, _
                  ByRef sampInterval As Double, _
                  ByRef nomSamples As Int32) As Int32
```

2.3.10 AcqrsD1_calibrate

Purpose

Performs an auto-calibration of the instrument.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

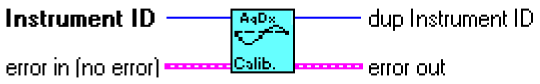
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_calibrate(ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Calibrate Instrument.vi
```



Visual Basic Representation

```
Calibrate (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_calibrate (ByVal instrumentID As Int32) As Int32
```

2.3.11 AcqrsD1_calibrateEx

Purpose

Performs a (partial) auto-calibration of the instrument.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
calType	ViInt32	= 0 calibrate the entire instrument = 1 calibrate only the current channel configuration = 2 calibrate external clock timing. Requires operation in External Clock (Continuous). = 3 calibrate only at the current frequency (12-bit-FAMILY, only) = 4 fast calibration for current settings only
modifier	ViInt32	For calType = 0,1, or 2: Currently unused, set to "0" For calType = 3 or 4, 0 = calibrate for all channels n = calibrate for channel "n"
flags	ViInt32	Currently unused, set to "0"

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

Calling this function with **calType** = 0 is equivalent to calling **AcqrsD1_calibrate**.

Calibrating with **calType** = 1 reduces the calibration time in digitizers with many possible channel combinations, e.g. the DC271. However, the user must keep track of which channel combinations were calibrated, and request another such partial calibration when changing the channel configuration with the function **AcqrsD1_configChannelCombination**.

Calibrating with **calType** = 2 can only be done if the external input frequency is appropriately high. See the discussion in the **Programmer's Guide** section 3.12.2, **External Clock (Continuous)**. If the calibration cannot be done an error code will be returned. It is not applicable for AP240 Signal Analyzer Platforms.

Calibrating with **calType** = 3 is for 12-bit digitizers only and is needed to support the HRes SR functionality. For best results it, or the longer full calibration, should be called after a change of sampling rate.

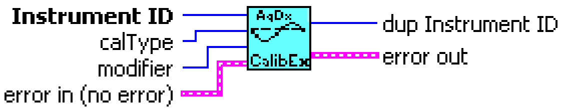
Calibrating with **calType** = 4 is for DC135, DC140, DC211A, DC241A, and DC271A models. A new calibration should be done if the **AcqrsD1_configChannelCombination** parameters or any of the following **AcqrsD1_configVertical** parameters are changed: fullScale, coupling (impedance), bandwidth, channel. This calibration will be much faster than the **calType** = 0 case for models with more than one impedance setting.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_calibrateEx(ViSession instrumentID,
                     ViInt32 calType, ViInt32 modifier, ViInt32
                     flags);
```

LabVIEW Representation

```
AqDx CalibrateEx Instrument.vi
```



Visual Basic Representation

```
CalibrateEx (ByVal instrumentID As Long, _
             ByVal calType As Long, _
             ByVal modifier As Long, _
             ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_calibrateEx (ByVal instrumentID As Int32, _
             ByVal calType As Int32, _
             ByVal modifier As Int32, _
             ByVal flags As Int32) As Int32
```

2.3.12 AcqrsD1_closeAll

Purpose

Closes all instruments in preparation for closing the application.

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function should be the last call to the driver, before closing an application. Make sure to stop *all* instruments beforehand.

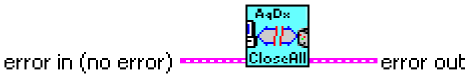
If this function is not called, closing the application might crash the computer in some situations, particularly in multi-threaded applications.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_closeAll(void);
```

LabVIEW Representation

```
AqDx Close All Instruments.vi
```



Visual Basic Representation

```
CloseAll ( ) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_closeAll ( ) As Int32
```

2.3.13 AcqrsD1_configAvgConfig

Purpose

Configures a parameter for averager/analyzer operation.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channelNbr	ViInt32	Channel number. A value = 0 will be treated as =1 for compatibility.
parameterString	ViString	Character string defining the requested parameter. See below for the list of accepted strings.
value	ViAddr	Value to set. ViAddr resolves to void* in C/C++. The user must allocate the appropriate variable type (as listed below), set it to the requested value and supply its address as 'value'.

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Accepted Parameter Strings

Parameter String	Data Type	Description
"DitherRange"	ViInt32	Range of offset dithering, in ADC LSB's. May assume values v = 0, 1…15. The offset is dithered over the range [-v, + v] in steps of ~1/8 LSB. For Averagers ONLY.
"GateType"	ViInt32	For AP240/AP235 Analyzers ONLY. 0 = No gates 1 = User Gates 2 = Threshold Gates
"HistoTDCEnable"	ViInt32	May assume 0 for not enabled and 1 to enable the simple TDC mode for the channel
"InvertData"	ViInt32	May assume 0 (no inversion) and 1 (invert data, 1's complement).
"NbrSamples"	ViInt32	Number of data samples per waveform segment. May assume values between 16 or 32 and the available memory length, in multiples of 16 (32) as explained below.
"NbrSegments"	ViInt32	Number of waveform segments to acquire. May assume values between 1 and 8192.
"NbrWaveforms"	ViInt32	Number of waveforms to average before going to next segment. May assume values between 1 and 65535 (64K – 1). For Averagers ONLY.
"NbrRoundRobins"	ViInt32	Number of times to perform the full segment cycle during data accumulation. For AP240/AP235 Averagers ONLY.
"P1Control"	ViInt32	May assume 0 = not enabled For AP240/AP235 Averagers ONLY. 1 = addSub channel 1 2 = addSub channel 2 3 = addSub channel 1 + 2 4 = average trigger enable 5 = start veto enable 6 = average (out) For AP240/AP235 SSR ONLY. 1 = Timestamp reset enable

Parameter String	Data Type	Description
"P2Control"	ViInt32	May assume 0 = not enabled For AP240/AP235 Averagers ONLY. 1 = addSub channel 1 2 = addSub channel 2 3 = addSub channel 1 + 2 4 = average trigger enable 5 = start veto enable 6 = average (out) For AP240/AP235 SSR ONLY. 1 = Timestamp reset enable
"PostSamples"	ViInt32	For AP240/AP235 Analyzers in Threshold Gate mode. Used to guarantee a number of samples before the first one satisfying the threshold condition. The meaningful values are 0,4,8,12,16. Other values will be rounded up or adapted appropriately.
"PreSamples"	ViInt32	For AP240/AP235 Analyzers in Threshold Gate mode. Used to guarantee a number of samples after the last one satisfying the threshold condition. The meaningful values are 0,4,8,12,16. Other values will be rounded up or adapted appropriately.
"StartVetoEnable"	ViInt32	For AP100/AP200 Averagers ONLY May assume 0 = for trigger enable functionality and 1 = use high state of I/O signal to allow the average accumulation to start. Must be used in conjunction with **AcqrsD1_configControlIO.**
"StartDelay"	ViInt32	Start delay in samples. May assume values between 0 and 33554400(16777216) in steps of 16 (32) as explained below. The limit is StepSize*(1024*1024-1).
"StopDelay "	ViInt32	Stop delay in samples. May assume values between 0 and 2097120(1048560) in steps of of 16 (32) as explained below. The limit is StepSize*(64*1024-1)
"TrigAlways"	ViInt32	May assume 0 (no trigger output) and 1 (trigger output on), in the case of no acquisition.
"TrigResync"	ViInt32	May assume 0 (no resync), 1 (resync) and 2 (free run)
"ThresholdEnable"	ViInt32	May assume 0 (no threshold) and 1 (threshold enabled). For Averagers ONLY.
"Threshold"	ViReal64	Value in Volts of the threshold for Noise Supressed Averaging or for SSR Threshold Gates.
"NoiseBaseEnable"	ViInt32	May assume 0 (no base subtraction) and 1 (base subtraction enabled). It can only be enabled if the threshold is enabled. For Averagers ONLY.
"NoiseBase"	ViReal64	Value in Volts of the value to be added in Noise Supressed Averaging. For Averagers ONLY.
"StartDeltaNegPeak"	ViInt32	Negative excursion needed before searching for negative peak. May assume values between 1 and 0xff. For Analyzers ONLY.
"StartDeltaPosPeak"	ViInt32	Positive excursion needed before searching for positive peak. May assume values between 1 and 0xff. For Analyzers ONLY.
"ValidDeltaNegPeak"	ViInt32	Positive excursion needed to validate a negative peak. May assume values between 1 and 0xff. For ONLY.
"ValidDeltaPosPeak"	ViInt32	Negative excursion needed to validate a positive peak. May assume values between 1 and 0xff. For AP101/AP201 ONLY.
"TriggerTimeout"	ViInt32	Trigger timeout in units of 30 ns in the range [0,2^{32} - 1]. A value of 0 means that no trigger will be generated and no *Prepare for Trigger* signal will be needed. For AP101/AP201 ONLY.

Parameter String	Data Type	Description
"TdcMinTOT"	ViInt32	The desired minimum width of a peak in the waveform; It can take on a value (n) from 1 to 4. A peak is accepted if there are at least n consecutive data samples above the Threshold. For SimpleTDC mode ONLY.
"TdcHistogramIncrement"	ViInt32	The desired increment to be applied for each entry; 1 means increment by 1, 2 means increment by the ADCvalue – NoiseBase. For SimpleTDC mode ONLY.

Discussion

The "TrigResync" values 0 and 1 require a valid trigger, while 2 requires no trigger (useful for background acquisition).

Set NbrWaveforms to 1 and NbrRoundRobins to n order to enable the round-robin segment acquisition mode with n triggers for each segment.

The channelNbr is used to designate the channel number for those parameters whose values can be different for the two channels of an AP240/AP235 in dual-channel mode. These parameters are indicated in **bold** in the list above.

The granularity for "NbrSamples","StartDelay", and "StopDelay" is 16 for the AP100/AP101 and the AP240/AP235 in Dual-Channel mode and 32 for the AP200/AP201 and the AP240/AP235 in Single-Channel mode.

If P1Control and/or P2Control are enabled for the Add/Subtract mode then the data will be added if the signal, or the or of both signals, is in the high state. The same rule holds if they are used for trigger enable.

The P1Control/P2Control "average (out)" signal goes high after the first trigger is accepted for an average and drops back down when the last trigger's acquition is complete.

Example

```
long channelNbr = 0, dither = 8;
AcqrsD1_configAvgConfig(ID, channelNbr, "DitherRange", &dither);
```

This function sets the dithering range to ± 8 LSB’s.

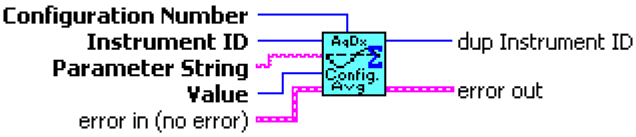
Note that this function takes the **address**, not the value of the parameter to be set.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configAvgConfig(ViSession instrumentID,
                     ViInt32 channelNbr, ViString parameterString,
                     ViAddr value);
```

LabVIEW Representation

```
AqDx Configure Averager Settings.vi
```



Visual Basic Representation

```
ConfigAvgConfig (ByVal instrumentID As Long, _
                 ByVal channelNbr As Long, _
                 ByVal parameterString As String, _
                 value As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configAvgConfig (ByVal instrumentID As Int32, _
                 ByVal channelNbr As Int32, _
                 ByVal parameterString As String, _
                 ByRef value As Int32) As Int32
or

AcqrsD1_ConfigAvgConfig (ByVal instrumentID As Int32, _
                 ByVal channelNbr As Int32, _
                 ByVal parameterString As String, _
                 ByRef value As Double) As Int32
```

2.3.14 AcqrsD1_configChannelCombination

Purpose

Configures how many converters are to be used for which channels. This routine is for use with some DC271-FAMILY instruments and the AP240/AP235 Signal Analyzer platforms.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
nbrConvertersPerChannel	ViInt32	= 1 all channels use 1 converter each (default) = 2 half of the channels use 2 converters each = 4 1/4 of the channels use 4 converters each
usedChannels	ViInt32	bit-field indicating which channels are used. See discussion below

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The acceptable values for 'usedChannels' depend on 'nbrConvertersPerChannel' and on the number of available channels in the digitizer:

A) If 'nbrConvertersPerChannel' = 1, 'usedChannels' must reflect the fact that ALL channels are available for use. It accepts a single value for a given digitizer:

'usedChannels'	= 0x00000001	if the digitizer has 1 channel
	= 0x00000003	if the digitizer has 2 channels
	= 0x0000000f	if the digitizer has 4 channels

B) If 'nbrConvertersPerChannel' = 2, 'usedChannels' must reflect the fact that only half of the channels may be used:

'usedChannels'	= 0x00000001	use channel 1 on a 2-channel digitizer
	= 0x00000002	use channel 2 on a 2-channel digitizer
	= 0x00000003	use channels 1+2 on a 4-channel digitizer
	= 0x00000005	use channels 1+3 on a 4-channel digitizer
	= 0x00000009	use channels 1+4 on a 4-channel digitizer
	= 0x00000006	use channels 2+3 on a 4-channel digitizer
	= 0x0000000a	use channels 2+4 on a 4-channel digitizer
	= 0x0000000c	use channels 3+4 on a 4-channel digitizer

C) If 'nbrConvertersPerChannel' = 4, 'usedChannels' must reflect the fact that only 1 of the channels may be used:

'usedChannels'	= 0x00000001	use channel 1 on a 4-channel digitizer
	= 0x00000002	use channel 2 on a 4-channel digitizer
	= 0x00000004	use channel 3 on a 4-channel digitizer
	= 0x00000008	use channel 4 on a 4-channel digitizer

NOTE: Digitizers which don't support channel combination, always use the default 'nbrConvertersPerChannel' = 1, and the single possible value of 'usedChannels'

NOTE: Changing the channel combination doesn't change the names of the channels; they are always the same.

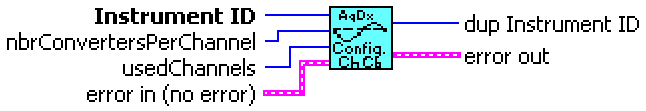
NOTE: If digitizers are combined with ASBus, the channel combination applies equally to all participating digitizers.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configChannelCombination(
                     ViSession instrumentID,
                     ViInt32 nbrConvertersPerChannel,
                     ViInt32 usedChannels);
```

LabVIEW Representation

```
AqDx Configure Channel Combination.vi
```



Visual Basic Representation

```
ConfigChannelCombination (ByVal instrumentID As Long, _
                       ByVal nbrConvertersPerChannel As Long, _
                       ByVal usedChannels As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configChannelCombination (ByVal instrumentID As Int32, _
                       ByVal nbrConvertersPerChannel As Int32, _
                       ByVal usedChannels As Int32) As Int32
```

2.3.15 AcqrsD1_configControlIO

Purpose

Configures a ControlIO connector. (For DC271-FAMILY/AP-FAMILY/ 12-bit-FAMILY only)

Parameters

Input

Name	Type	Description
InstrumentID	ViSession	Instrument identifier
Connector	ViInt32	Connector Number 1 = Front Panel I/O A (MMCX connector) 2 = Front Panel I/O B (MMCX connector) 9 = Front Panel Trigger Out (MMCX connector) 11 = PXI Bus 10 MHz (DC135/DC140/DC211/ DC211A/DC241/DC241A/DC271/DC271A) 12 = PXI Bus Star Trigger (same models as above)
Signal	ViInt32	The accepted values depend on the type of connector See the table below for details.
qualifier1	ViInt32	The accepted values depend on the type of connector See the table below for details.
qualifier2	ViReal64	If trigger veto functionality is available (AP101/AP201 only), accepts values between 30 ns and 1.0 sec. The trigger veto values given will be rounded off to steps of 33 ns. A value of 0.0 means that no holdoff is required and no *Prepare for Trigger* signal will be needed.

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Accepted Values of *signal* vs. Connector Type

Connector Type	Possible Values of *signal* and *qualifierX*
Front Panel I/O	0 = Disable **Inputs:** 6 = (Level) Enable trigger input (for Digitizers) If one of the two I/O connectors is set to this value then a high level must be present before an edge can be accepted. If both I/O connectors are set to this value, they both must be high before the trigger edge can be accepted. 6 = (Level) Enable trigger input or Start Veto (for AP100/AP200 Averagers) see **AcqrsD1_configAvgConfig** for more 8 = *Prepare for Trigger* signal present on this connector. *qualifier2* gives the desired holdoff in time. **Outputs:** 19 = (Clock) 10 MHz reference clock 20 = (Pulse) Acquisition skips to next segment (in sequence acquisition mode) input (Not for AP240/AP235 Signal Analyzers). 21 = (Level) Acquisition is active 22 = (Level) Trigger is armed (ready) The values of *qualifier1* and *qualifier2* are not used

Connector Type	Possible Values of *signal* and *qualifierX*
Front Panel Trigger Out	The value of *signal* is interpreted as a signal offset in mV. E.g. *signal* = -500 offsets the output signal by –500 mV. The accepted range of *signal* is [-2500,2500], i.e. ± 2.5 V with a resolution of ~20 mV. The value of *qualifier1* controls if the trigger output is resynchronized to the clock or maintains a precise timing relation to the trigger input. *qualifier1*= 0 (default): Non-resynchronized *qualifier1*= 1 : Resynchronized to sampling clock
PXI Bus 10 MHz	0 = Disable 1 = Enable Replaces the internal 10 MHz reference clock with the 10 MHz clock on the PXI rear panel connector
PXI Bus Star Trigger	0 = Disable 1 = Use PXI Bus Star Trigger as Trigger Input 2 = Use PXI Bus Star Trigger for Trigger Output **Note:** When using this connector as Trigger Input, you also must set the trigger source in *sourcePattern* in the function **AcqrsD1_configTrigClass** to External Trigger2!

Discussion

ControlIO connectors are front panel IO connectors for special purpose control functions of the digitizer. Typical examples are user-controlled acquisition control (start/stop/skip) or control output signals such as ‘acquisition ready’ or ‘trigger ready’.

The connector numbers are limited to the allowed values. To find out which connectors are supported by a given module, use the query function **AcqrsD1_getControlIO**.

The variable *signal* specifies the (programmable) use of the specified connector.

In order to set I/O A as a ‘Enable Trigger’ input and the I/O B as a 10 MHz reference output, use the function calls

```
AcqrsD1_configControlIO(instrID, 1,  6, 0, 0.0);
AcqrsD1_configControlIO(instrID, 2, 19, 0, 0.0);
```

In order to obtain a signal offset of +1.5 V on the Trigger Output, use the call

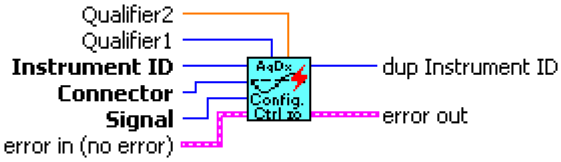
```
AcqrsD1_configControlIO(instrID, 9, 1500, 0, 0.0);
```

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configControlIO(ViSession instrumentID,
                        ViInt32 connector, ViInt32 signal,
                        ViInt32 qualifier1, ViReal64 qualifier2);
```

LabVIEW Representation

AqDx Configure Control IO Connectors.vi



Visual Basic Representation

```
ConfigControlIO (ByVal instrumentID As Long, _
                 ByVal connector As Long, _
                 ByVal signal As Long, _
                 ByVal qualifier1 As Long, _
                 ByVal qualifier2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configControlIO (ByVal instrumentID As Int32, _
                 ByVal connector As Int32, _
                 ByVal signal As Int32, _
                 ByVal qualifier1 As Int32, _
                 ByVal qualifier2 As Double) As Int32
```

2.3.16 AcqrsD1_configExtClock

Purpose

Configures the external clock of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
clockType	ViInt32	= 0 Internal Clock (default at start-up) = 1 External Clock, continuously running = 2 External Reference (10 MHz) = 4 External Clock, with start/stop sequence
inputThreshold	ViReal64	Input threshold for external clock or reference in mV
delayNbrSamples	ViInt32	Number of samples to acquire after trigger (for digitizers using 'clockType' = 1 only!)
inputFrequency	ViReal64	The input frequency of the external clock, for clockType = 1 only
sampFrequency	ViReal64	The desired Sampling Frequency, for clockType = 1 only

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

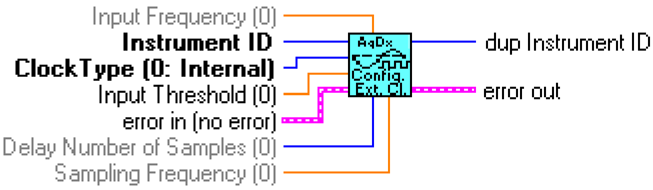
When **clockType** is set to 1 or 4, the parameters of the function **AcqrsD1_configHorizontal** are ignored! Please refer to your product User Manual, for the conditions on the clock signals, and to the **Programmer's Guide** section 3.12**, External Clock**, for the setup parameters and the theory of operation.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configExtClock(ViSession instrumentID,
                     ViInt32 clockType, ViReal64 inputThreshold,
                     ViInt32 delayNbrSamples, ViReal64 inputFrequency,
                     ViReal64 sampFrequency);
```

LabVIEW Representation

```
AqDx Configure External Clock.vi
```



Visual Basic Representation

```
ConfigExtClock (ByVal instrumentID As Long, _
                ByVal clockType As Long, _
                ByVal inputThreshold As Double, _
                ByVal delayNbrSamples As Long, _
                ByVal inputFrequency As Double, _
                ByVal sampFrequency As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configExtClock (ByVal instrumentID As Int32, _
                ByVal clockType As Int32, _
                ByVal inputThreshold As Double, _
                ByVal delayNbrSamples As Int32, _
                ByVal inputFrequency As Double, _
                ByVal sampFrequency As Double) As Int32
```

2.3.17 AcqrsD1_configFCounter

Purpose

Configures a frequency counter measurement

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
signalChannel	ViInt32	Signal input channel
type	ViInt32	Type of measurement = 0 Frequency (default) = 1 Period (1/frequency) = 2 Totalize by Time = 3 Totalize by Gate
targetValue	ViReal64	User-supplied estimate of the expected value, may be 0.0 if no estimate is available.
apertureTime	ViReal64	Time in sec, during which the measurement is executed, see discussion below.
reserved	ViReal64	Currently ignored
flags	ViInt32	Currently ignored

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The Frequency mode (type = 0) measures the frequency of the signal applied to the selected ‘signalChannel’ during the aperture time. The default value of ‘apertureTime’ is 0.001 sec and can be set to any value between 0.001 and 1000.0 seconds. A longer aperture time may improve the measurement accuracy, if the (externally applied) reference clock has a high accuracy and/or if the signal slew rate is low.
The ‘targetValue’ is a user-supplied estimated of the expected result, and helps in choosing the optimal measurement conditions. If the supplied value is < 1000.0, and > 0.0, then the instrument will not use the HF trigger mode to divide the input frequency. Otherwise, it divides it by 4 in order to obtain a larger frequency range.

The Period mode (type = 1) is equal to the frequency mode, but the function **AcqrsD1_readFCounter** returns the inverse of the measured frequency. If the ‘targetValue’ is < 0.001 (1 ms), then the instrument will not use the HF trigger mode, otherwise it does.

The Totalize by Time mode (type = 2) counts the number of pulses in the signal applied to the selected ‘signalChannel’ during the time defined by ‘apertureTime’. The ‘targetValue’ is ignored.

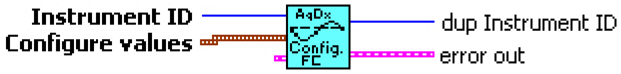
The Totalize by Gate mode (type = 3) counts the number of pulses in the signal applied to the selected ‘signalChannel’ during the time defined by signal at the I/O A or I/O B inputs on the front panel. The gate is open while the signal is high, and closed while the signal is low (if no signal is connected, counting will be enabled, since there is an internal pull-up resistor). The gate may be opened/closed several times during the measurement. The measurement must be terminated with the function **AcqrsD1_stopAcquisition**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configFCounter(ViSession instrumentID,
         ViInt32 signalChannel, ViInt32 type, ViReal64 targetValue,
         ViReal64 apertureTime,ViReal64 reserved, ViInt32 flags);
```

LabVIEW Representation

AqDx Configure FCounter.vi



Visual Basic Representation

```
ConfigFCounter (ByVal instrumentID As Long, _
                ByVal signalChannel As Long, _
                ByVal type As Long, _
                ByVal targetValue As Double, _
                ByVal apertureTime As Double, _
                ByVal reserved As Double, _
                ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configFCounter (ByVal instrumentID As Int32, _
                ByVal signalChannel As Int32, _
                ByVal type As Int32, _
                ByVal targetValue As Double, _
                ByVal apertureTime As Double, _
                ByVal reserved As Double, _
                ByVal flags As Int32) As Int32
```

2.3.18 AcqrsD1_configHorizontal

Purpose

Configures the horizontal control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
sampInterval	ViReal64	Sampling interval in seconds
delayTime	ViReal64	Trigger delay time in seconds, with respect to the beginning of the record. A positive number corresponds to a trigger *before* the beginning of the record (post-trigger recording). A negative number corresponds to pre-trigger recording. It can't be less than -(sampInterval * nbrSamples), which corresponds to 100% pre-trigger.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

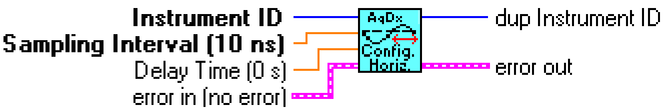
Refer to the **Programmer's Guide** section 3.9, **Trigger Delay and Horizontal Waveform Position**, for a detailed discussion of the value **delayTime**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configHorizontal(ViSession instrumentID,
                      ViReal64 sampInterval, ViReal64 delayTime);
```

LabVIEW Representation

```
AqDx Configure Horizontal Settings.vi
```



Visual Basic Representation

```
ConfigHorizontal (ByVal instrumentID As Long, _
                  ByVal sampInterval As Double, _
                  ByVal delayTime As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configHorizontal (ByVal instrumentID As Int32, _
                  ByVal sampInterval As Double, _
                  ByVal delayTime As Double) As Int32
```

2.3.19 AcqrsD1_configLogicDevice

Purpose

NOTE: This function now needs to be used only by ETS and VxWorks users to specify the filePath for FPGA .bit files. Otherwise it should no longer have to be used

Configures (programs) on-board logic devices, such as user-programmable FPGA's.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
deviceName	ViChar [ ]	Identifies which device to program In the AP100/AP101, AP235/AP240, and the 12-bit modules this string must be "Block1Devx", with x = 1 or 2. In the AP200/AP201, it must be "Block1Devx", with x = 1, 2, 3 or 4. When clearing the FPGA' s, the string must be "Block1DevAll".
filePathName	ViChar [ ]	File path and file name
modifier	ViInt32	Modifier, may be: 0 = program logic device with data in the file "filePathName" 1 = clear the logic device 2 = set path where FPGA .bit files can be found

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

With modifier = 1 in VxWorks systems, the filePathName must point to a directory containing the FPGA configuration files with extension '.bit'

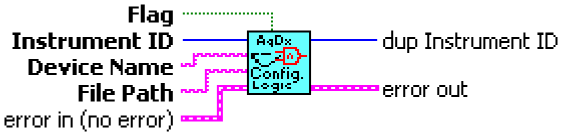
With modifier = 0, the filePathName must point to an FPGA configuration file with extension '.bit', e.g. "D:\Averagers\FPGA\AP100DefaultFPGA1.bit".
For more details on programming on-board logic devices, please refer to the **Programmer's Guide** sections 3.2, **Device Initialization** and 3.3, **Device Configuration**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configLogicDevice(ViSession instrumentID,
                        ViChar deviceName[], ViChar filePathName[],
                        ViInt32 modifier);
```

LabVIEW Representation

AqDx Configure Logic Device.vi



Visual Basic Representation

```
ConfigLogicDevice (ByVal instrumentID As Long, _
                   ByVal deviceName As String, _
                   ByVal filePathName As String, _
                   ByVal modifier As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configLogicDevice (ByVal instrumentID As Int32, _
                   ByVal deviceName As String, _
                   ByVal filePathName As String, _
                   ByVal modifier As Int32) As Int32
```

2.3.20 AcqrsD1_configMemory

Purpose

Configures the memory control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
nbrSamples	ViInt32	Nominal number of samples to record (per segment!)
nbrSegments	ViInt32	Number of segments to acquire. 1 corresponds to the normal single-trace acquisition mode.

Return Value

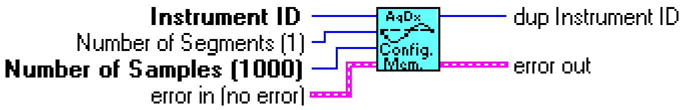
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configMemory(ViSession instrumentID,
                     ViInt32 nbrSamples, ViInt32 nbrSegments);
```

LabVIEW Representation

```
AqDx Configure Memory Settings.vi
```



Visual Basic Representation

```
ConfigMemory (ByVal instrumentID As Long, _
              ByVal nbrSamples As Long, _
              ByVal nbrSegments As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configMemory (ByVal instrumentID As Int32, _
              ByVal nbrSamples As Int32, _
              ByVal nbrSegments As Int32) As Int32
```

2.3.21 AcqrsD1_configMode

Purpose

Configures the operational mode of Averagers and Analyzers. It doesn't apply to digitizers.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
mode	ViInt32	0 = normal data acquisition 2 = averaging mode (only in real-time averagers) 3 = buffered data acquisition (only in AP101/AP201 analyzers) 6 = frequency counter mode 7 = AP235/AP240-SSR mode
modifier	ViInt32	Currently not used, set to 0
flags	ViInt32	If 'mode' = 0, this variable can take these values: 0 = 'normal' (default value) 1 = 'Start on Trigger' mode 2 = 'Sequence Wrap' mode If 'mode' = 2, this variable is not used (set to 0). For AP101/AP201 units, if 'mode' = 3, this variable can take these values: 0 = acquire into 1st memory bank 1 = acquire into 2nd memory bank

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

Most digitizers only permit the default *mode* = 0. Real-time averagers support the normal data acquisition mode (0) and the averager mode (2). The analyzers (digitizers with buffered acquisition memory) (AP101/AP201 and AP235/AP240 with SSR) support both the normal data acquisition mode (0) *and* the buffered mode (3).

The normal data acquisition mode (0) supports the following submodes:

- flags = 0: normal digitizer mode

- flags = 1: 'StartOnTrigger' mode, whereby data recording only begins after the receipt of a valid trigger. For details, see **Programmer's Guide** section 3.14, **Special Operating Modes**.

- flags = 2: 'Sequence Wrap' mode, whereby a multi-segment acquisition (with 'nbrSegments' > 1, when configured with the function **AcqrsD1_configMemory**), does not stop after 'nbrSegments', but *wraps around* to zero, indefinitely. Thus, such acquisitions must be stopped with the function **AcqrsD1_stopAcquisition** at the appropriate moment. The digitizer memory then contains the last (nbrSegments-1) waveform segments. For details, see **Programmer's Guide** section 3.14, **Special Operating Modes**.

The averaging mode (2) has the following differences from the default mode (0):

- The function **AcqrsD1_acquire()**: In mode 0, it starts a normal waveform acquisition, whereas in mode 2, it makes the instrument run as a real-time averager.

- The function **AcqrsD1_readData()** with **dataType = ReadReal64**: In mode 0, it returns the last acquired waveform, whereas in mode 2, it returns the averaged waveform (in Volts).

The buffered data acquisition mode (3) and the SSR mode (7) have the following differences from the default mode (0):

- The function **AcqrsD1_acquire()**: In mode 0, it starts a normal waveform acquisition, whereas in modes 3 or 7, it starts an acquisition into the next memory bank or a special memory bank, as defined by *flags*.

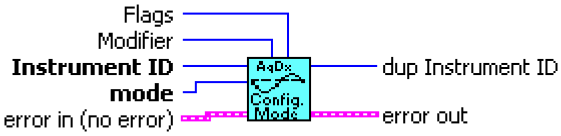
- The functions **AcqrsD1_readData()**: In mode 0, they return the last acquired waveform from the normal acquisition memory, whereas in mode 3, they return data from a memory bank (opposite to what is defined by *flags*).

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configMode(ViSession instrumentID,
                     ViInt32 mode, ViInt32 modifier, ViInt32 flags);
```

LabVIEW Representation

```
AqDx Configure Operation Mode.vi
```



Visual Basic Representation

```
ConfigMode (ByVal instrumentID As Long, _
            ByVal mode as Long, _
            ByVal modifier As Long, _
            ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configMode (ByVal instrumentID As Int32, _
            ByVal mode as Int32, _
            ByVal modifier As Int32, _
            ByVal flags As Int32) As Int32
```

2.3.22 AcqrsD1_configMultiInput

Purpose

Selects the active input when there are multiple inputs on a channel. It is useful for Averagers, Analyzers, and some digitizer models.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
input	ViInt32	= 0 set to input connection A = 1 set to input connection B

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

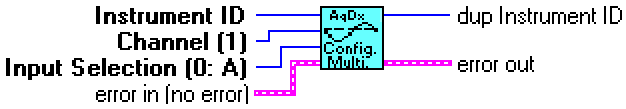
This function is only of use for instruments with an input-multiplexer (i.e. more than 1 input per digitizer, e.g. DP211). On the "normal" instruments with a single input per channel, this function may be ignored.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configMultiInput(ViSession instrumentID,
                      ViInt32 channel, ViInt32 input);
```

LabVIEW Representation

```
AqDx Configure Multiplexer Input.vi
```



Visual Basic Representation

```
ConfigMultiInput (ByVal instrumentID As Long, _
                  ByVal channel As Long, _
                  ByVal connection As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configMultiInput (ByVal instrumentID As Int32, _
                  ByVal channel As Int32, _
                  ByVal connection As Int32) As Int32
```

2.3.23 AcqrsD1_configSetupArray

Purpose

Sets the configuration for an array of configuration values. It is useful for Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
setupType	ViInt32	Type of setup. 0 = GateParameters
nbrSetupObj	ViInt32	Number of setup objects in the array
setupData	ViAddr	Pointer to an array containing the setup objects `ViAddr` resolves to `void*` in C/C++. The user must allocate the appropriate variable type and supply its address as 'setupData'.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

GateParameters

Name	Type	Description
GatePos	ViInt32	Start position of the gate (must be multiple of 4)
GateLength	ViInt32	Length of the gate (must be multiple of 4)

Discussion

The user has to take care to allocate sufficient memory for the setupData. nbrSetupObj should not be higher than what the allocated setupData holds.

The SSR option allows up to 4095 gate definitions. The AP101/AP201 analyzers are limited to 64 gate definitions.

Note: The driver contains a set of 4095(64) default AqGateParameters, defined as { {0,256} {256, 256} {512, 256} {768, 256} ... }.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configSetupArray(ViSession instrumentID,
                        ViInt32 channel,
                        ViInt32 setupType, ViInt32 nbrSetupObj,
                        ViAddr setupData);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
ConfigSetupArray (ByVal instrumentID As Long, _
                  ByVal channel As Long, _
                  ByVal setupType As Long, _
                  ByVal nbrSetupObj As Long, _
                  setupData As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configSetupArray (ByVal instrumentID As Int32, _
                  ByVal channel As Int32, _
                  ByVal setupType As Int32, _
                  ByVal nbrSetupObj As Int32, _
                  ByRef setupData As Int32) As Int32
```

2.3.24 AcqrsD1_configTrigClass

Purpose

Configures the trigger class control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
trigClass	ViInt32	= 0 edge trigger = 1 TV trigger (12-bit-FAMILY External only)
sourcePattern	ViInt32	= `0x000n0001` for Channel 1, = `0x000n0002` for Channel 2, = `0x000n0004` for Channel 3, = `0x000n0008` for Channel 4 etc. = `0x800n0000` for External Trigger 1, = `0x400n0000` for External Trigger 2 etc. where n is 0 for single instruments, or the module number for *MultiInstruments* (ASBus operation). See discussion below.
validatePattern	ViInt32	Currently ignored
HoldType	ViInt32	Currently ignored
holdValue1	ViReal64	Currently ignored
holdValue2	ViReal64	Currently ignored

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The number of internal (i.e. channel) or external trigger sources of the instrument can be retrieved with the **AcqrsD1_getInstrumentInfo** function.

For more details on the trigger source pattern in ASBus-connected MultiInstruments, please refer to the **Programmer's Guide** section 3.13.2, **Trigger Source Numbering with ASBus**.

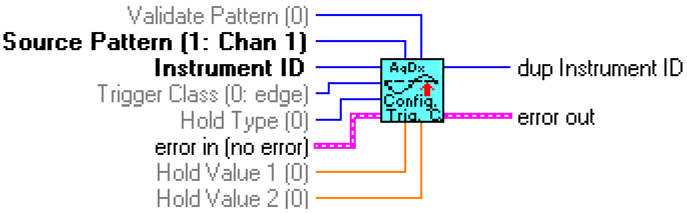
For configuring the TV trigger see **AcqrsD1_configTrigTV**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configTrigClass(ViSession instrumentID,
                     ViInt32 trigClass, ViInt32 sourcePattern,
                     ViInt32 validatePattern, ViInt32 holdType,
                     ViReal64 holdValue1, ViReal64 holdValue2);
```

LabVIEW Representation

AqDx Configure Trigger Class.vi



Visual Basic Representation

```
ConfigTrigClass (ByVal instrumentID As Long, _
                 ByVal trigClass As Long, _
                 ByVal sourcePattern As Long, _
                 ByVal validatePattern As Long, _
                 ByVal holdType As Long, _
                 ByVal holdValue1 As Double, _
                 ByVal holdValue2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configTrigClass (ByVal instrumentID As Int32, _
                 ByVal trigClass As Int32, _
                 ByVal sourcePattern As Int32, _
                 ByVal validatePattern As Int32, _
                 ByVal holdType As Int32, _
                 ByVal holdValue1 As Double, _
                 ByVal holdValue2 As Double) As Int32
```

2.3.25 AcqrsD1_configTrigSource

Purpose

Configures the trigger source control parameters for the specified trigger source (channel or External).

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	= 1...(Number of IntTrigSources) for internal sources = -1..-(Number of ExtTrigSources) for external sources See discussion below.
trigCoupling	ViInt32	= 0 DC = 1 AC = 2 HF Reject (if available) = 3 DC, 50 Ω (ext. trigger only, if available) = 4 AC, 50 Ω (ext. trigger only, if available)
trigSlope	ViInt32	= 0 Positive = 1 Negative = 2 out of Window = 3 into Window = 4 HF divide
trigLevel1	ViReal64	Trigger threshold in % of the vertical Full Scale of the channel, or in mV if using an External trigger source. See discussion below.
trigLevel2	ViReal64	Trigger threshold 2 (as above) for use when Window trigger is selected

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The number of internal (i.e. channel) or external trigger sources of the instrument can be retrieved with the **AcqrsD1_getInstrumentInfo** function.

The allowed range for the trigger threshold depends on the model and the channel chosen. See your product User Manual.

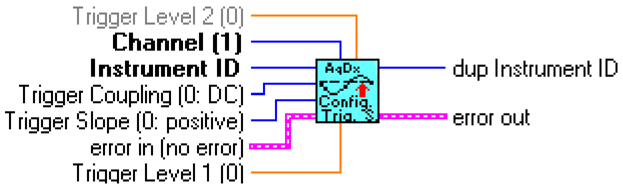
NOTE: Some of the possible states may be unavailable in some digitizers. In particular, the trigCoupling choices of ‘DC, 50 Ω’ and ‘AC, 50 Ω’ are only needed for modules that have both 50 Ω and 1 MΩ external input impedance possibilities.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configTrigSource(ViSession instrumentID,
                     ViInt32 channel, ViInt32 trigCoupling,
                     ViInt32 trigSlope, ViReal64 trigLevel1,
                     ViReal64 trigLevel2);
```

LabVIEW Representation

AqDx Configure Trigger Source.vi



Visual Basic Representation

```
ConfigTrigSource (ByVal instrumentID As Long, _
                  ByVal Channel As Long, _
                  ByVal trigCoupling As Long, _
                  ByVal trigSlope As Long, _
                  ByVal trigLevel1 As Double, _
                  ByVal trigLevel2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configTrigSource (ByVal instrumentID As Int32, _
                  ByVal Channel As Int32, _
                  ByVal trigCoupling As Int32, _
                  ByVal trigSlope As Int32, _
                  ByVal trigLevel1 As Double, _
                  ByVal trigLevel2 As Double) As Int32
```

2.3.26 AcqrsD1_configTrigTV

Purpose

Configures the TV trigger parameters (12-bit-FAMILY only).

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	= -1..-(Number of ExtTrigSources) for external sources See discussion below.
standard	ViInt32	= 0 625/50/2:1 (PAL or SECAM) = 2 525/60/2:1 (NTSC)
field	ViInt32	= 1 Field 1 - odd = 2 Field 2 - even
line	ViInt32	= line number, depends on the parameters above: For 'standard' = 625/50/2:1 = 1 to 313 for 'field' = 1 = 314 to 625 for 'field' = 2 For 'standard' = 525/60/2:1 = 1 to 263 for 'field' = 1 = 1 to 262 for 'field' = 2

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

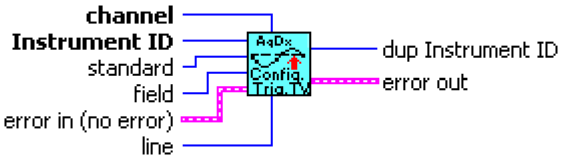
The number of internal (i.e. channel) or external trigger sources of the instrument can be retrieved with the **AcqrsD1_getInstrumentInfo** function.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configTrigTV (ViSession instrumentID,
                     ViInt32 channel, ViInt32 standard,
                     ViInt32 field, ViInt32 line);
```

LabVIEW Representation

AqDx Configure Trigger TV.vi



Visual Basic Representation

```
ConfigTrigTV (ByVal instrumentID As Long, _
                 ByVal Channel As Long, _
                 ByVal standard As Long, _
                 ByVal field As Long, _
                 ByVal line AS Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configTrigTV (ByVal instrumentID As Int32, _
                 ByVal Channel As Int32, _
                 ByVal standard As Int32, _
                 ByVal field As Int32, _
                 ByVal line AS Int32) As Int32
```

2.3.27 AcqrsD1_configVertical

Purpose

Configures the vertical control parameters for a specified channel of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan, or –1,… for the External Input
fullScale	ViReal64	in Volts
offset	ViReal64	in Volts
coupling	ViInt32	= 0 Ground (Averagers ONLY) = 1 DC, 1 MΩ = 2 AC, 1 MΩ = 3 DC, 50 Ω = 4 AC, 50 Ω
bandwidth	ViInt32	= 0 no bandwidth limit (default) = 1 bandwidth limit at 25 MHz = 2 bandwidth limit at 700 MHz = 3 bandwidth limit at 200 MHz = 4 bandwidth limit at 20 MHz = 5 bandwidth limit at 35 MHz

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

For the DC440 and DP310 the coupling input is used to select the signal input: DC, 50 Ω for the Standard input and AC, 50 Ω for the Direct HF input.

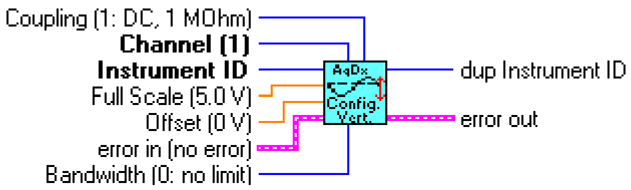
Some instruments have no bandwidth limiting capability. In this case, use **bandwidth** = 0. With **channel** = -1 this function can be used to set the Full Scale Range and the bandwidth limit of the external trigger for the DC271-FAMILY digitizers and the AP240/AP235 signal analyzer platforms. For the case of a DC271-FAMILY *MultiInstrument* using ASBus, the external triggers of the additional modules are numbered –3, -5, … following the principles given in the **Programmer's Guide** section 3.13.2, **Trigger Source Numbering with ASBus**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configVertical(ViSession instrumentID,
                     ViInt32 channel,ViReal64 fullScale,
                     ViReal64 offset, ViInt32 coupling,
                     ViInt32 bandwidth);
```

LabVIEW Representation

`AqDx Configure Vertical Settings.vi`



Visual Basic Representation

```
ConfigVertical (ByVal instrumentID As Long, ByVal Channel As Long, _
                ByVal fullScale As Double,  ByVal offset As Double, _
                ByVal coupling As Long, _
                ByVal bandwidth As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configVertical (ByVal instrumentID As Int32, _
                ByVal Channel As Int32, _
                ByVal fullScale As Double, _
                ByVal offset As Double, _
                ByVal coupling As Int32, _
                ByVal bandwidth As Int32) As Int32
```

2.3.28 AcqrsD1_errorMessage

Purpose

Translates an error code into a human readable form.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier can be VI_NULL
errorCode	ViStatus	Error code (returned by a function) to be translated

Output

Name	Type	Description
errorMessage	ViChar [ ]	Pointer to user-allocated string (minimum size 256) into which the error-message text is returned

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

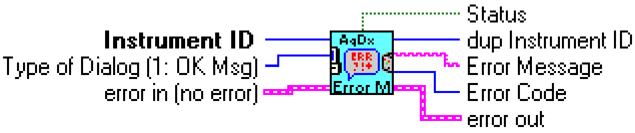
Discussion

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_errorMessage(ViSession instrumentID,
                        ViStatus errorCode, ViChar errorMessage[]);
```

LabVIEW Representation

```
AqDx Error Message.vi
```



Visual Basic Representation

```
errorMessage (ByVal instrumentID As Long, ByVal errorCode As Long, _
              ByVal errorMessage As String) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_errorMessage (ByVal instrumentID As Int32, _
              ByVal errorCode As Int32, _
              ByVal errorMessage As String) As Int32
```

2.3.29 AcqrsD1_forceTrig

Purpose

Forces a *manual* trigger. It should not be used for Averagers or Analyzers.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

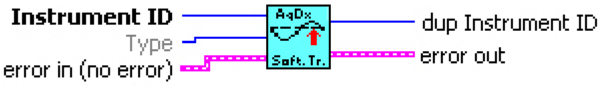
The function returns immediately after initiating an acquisition. One must therefore wait until the acquisition has terminated before reading the data, by checking the status with the **AcqrsD1_acqDone** function. If the external clock is enabled, and there is no clock signal applied to the device, **AcqrsD1_acqDone** will never return **done** = VI_TRUE. Consider using a timeout and calling **AcqrsD1_stopAcquisition** if it occurs. In multisegment mode, the current segment is acquired, the acquisition is terminated and the data and timestamps of subsequent segments are invalid.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_forceTrig(ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Software Trigger.vi
```



Visual Basic Representation

```
ForceTrig (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_forceTrig (ByVal instrumentID As Int32) As Int32
```

2.3.30 AcqrsD1_forceTrigEx

Purpose

Forces a *manual* trigger. It should not be used for Averagers or Analyzers.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
forceTrigType	ViInt32	= 0 Sends a software trigger to end the full acquisition = 1 Sends a single software trigger and generates the TrigOut hardware signal
modifier	ViInt32	Currently not used
flags	ViInt32	Currently not used

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The function returns immediately after initiating an acquisition. One must therefore wait until the acquisition has terminated before reading the data, by checking the status with the **AcqrsD1_acqDone** function. If the external clock is enabled, and there is no clock signal applied to the device, **AcqrsD1_acqDone** will never return **done** = VI_TRUE. Consider using a timeout and calling **AcqrsD1_stopAcquisition** if it occurs.

For forceTrigType = 0, the 'trigOut' Control IO will NOT generate a trigger output. This mode is equivalent to **AcqrsD1_forceTrig**. In multisegment mode, the current segment is acquired, the acquisition is terminated and the data and timestamps of subsequent segments are invalid.

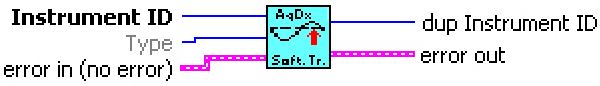
For forceTrigType = 1, 'trigOut' Control IO will generate a trigger output on each successful call. In multisegment mode, the acquisition advances to the next segment and then waits again for a trigger. If no valid triggers are provided to the device, the application must call AcqrsD1_forceTrigEx as many times as there are segments. Every acquired segment will be valid. This mode is only supported for single (i.e. non-ASBus-connected) digitizers (not Averagers or Analyzers).

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_forceTrigEx(ViSession instrumentID ,
                 ViInt32 forceTrigType, ViInt32 modifier, ViInt32 flags);
```

LabVIEW Representation

```
AqDx Software Trigger.vi
```



Visual Basic Representation

```
ForceTrigEx (ByVal instrumentID As Long, _
             ByVal forceTrigType as Long, _
             ByVal modifier As Long, _
             ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_forceTrigEx (ByVal instrumentID As Int32, _
             ByVal forceTrigType as Int32, _
             ByVal modifier As Int32, _
             ByVal flags As Int32) As Int32
```

2.3.31 AcqrsD1_getAvgConfig

Purpose

Returns an attribute from the analyzer/averager configuration ***channelNbr***.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channelNbr	ViInt32	Channel number for use with AP240/AP235 dual-channel mode. A value = 0 will be treated as =1 for compatibility.
parameterString	ViString	Character string defining the requested parameter. See **AcqrsD1_configAvgConfig** for the list of accepted strings.

Output

Name	Type	Description
value	ViAddr	Requested information value. `ViAddr` resolves to `void*` in C/C++. The user must allocate the appropriate variable type (as listed under **AcqrsD1_configAvgConfig**) and supply its address as 'value'.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

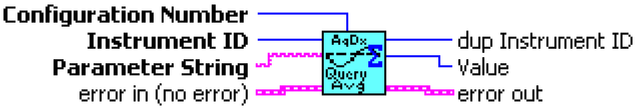
See remarks under **AcqrsD1_configAvgConfig**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getAvgConfig(ViSession instrumentID,
                     ViInt32 channelNbr, ViString parameterString,
                     ViAddr value);
```

LabVIEW Representation

```
AqDx Query Averager Settings.vi
```



Visual Basic Representation

```
GetAvgConfig (ByVal instrumentID As Long, _
              ByVal channelNbr As Long, _
              ByVal parameterString As String, _
              value as Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getAvgConfig (ByVal instrumentID As Int32, _
              ByVal channelNbr As Int32, _
              ByVal parameterString As String, _
              ByRef value as Int32) As Int32
```

or

```
AcqrsD1_getAvgConfig (ByVal instrumentID As Int32, _
              ByVal channelNbr As Int32, _
              ByVal parameterString As String, _
              ByRef value as Double) As Int32
```

2.3.32 AcqrsD1_getChannelCombination

Purpose

Returns the current channel combination parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nbrConvertersPer Channel	ViInt32	= 1 all channels use 1 converter each (default) = 2 half of the channels use 2 converters each = 4 1/4 of the channels use 4 converters each
usedChannels	ViInt32	bit-field indicating which channels are used. See discussion below

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

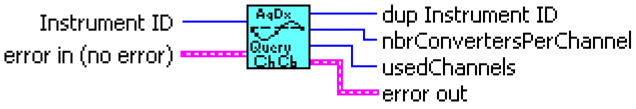
See remarks under **AcqrsD1_configChannelCombination**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getChannelCombination(
                      ViSession instrumentID,
                      ViInt32* nbrConvertersPerChannel,
                      ViInt32* usedChannels);
```

LabVIEW Representation

```
AqDx Query Channel Combination.vi
```



Visual Basic Representation

```
GetChannelCombination (ByVal instrumentID As Long, _
            nbrConvertersPerChannel As Long, _
            usedChannels As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getChannelCombination (ByVal instrumentID As Int32, _
            ByRef nbrConvertersPerChannel As Int32, _
            ByRef usedChannels As Int32) As Int32
```

2.3.33 AcqrsD1_getControlIO

Purpose

Returns the configuration of a ControlIO connector. (For DC271-FAMILY/AP-FAMILY/12-bit-FAMILY only)

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
connector	ViInt32	Connector Number 1 = Front Panel I/O A (MMCX connector) 2 = Front Panel I/O B (MMCX connector) 9 = Front Panel Trigger Out (MMCX connector)
signal	ViInt32	Indicates the current use of the specified connector 0 = Disabled, 6 = Enable trigger etc. For a detailed list, see the description of **`AcqrsD1_configControlIO`**
qualifier1	ViInt32	The returned values depend on the type of connector, see the discussion in **`AcqrsD1_configControlIO`**
qualifier2	ViReal64	The returned values depend on the module, see the discussion in **`AcqrsD1_configControlIO`**

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

ControlIO connectors are front panel IO connectors for special purpose control functions of the digitizer. Typical examples are user-controlled acquisition control (trigger enable) or control output signals such as ’10 MHz reference’ or ‘trigger ready’.

The connector numbers are limited to 0 and the supported values. To find out which connectors are supported by a given module, use this function with connector = 0:

```
AcqrsD1_getControlIO(instrID, 0, &ctrlIOPattern, NULL, NULL);
```

In this case, the returned value of *signal* is the bit-coded list of the *connectors* that are available in the digitizer. E.g. If the connectors 1 (I/O A), 2 (I/O B) and 9 (TrigOut) are present, the bits 1, 2 and 9 of *signal* are set, where bit 0 is the LSbit and 31 is the MSbit. Thus, the low order 16 bits of *signal* (or *ctrlIOPattern* in the example above) would be equal to 0x206.

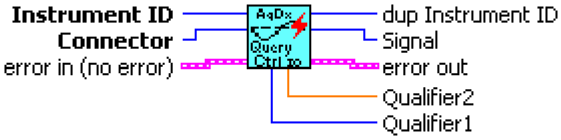
The DC271-FAMILY, AP-FAMILY, and 12-bit-FAMILY cards support the connectors 1 (front panel I/O A MMCX coax), 2 (front panel I/O B MMCX coax) and 9 (front panel Trig Out MMCX coax).

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getControlIO(ViSession instrumentID,
                     ViInt32 connector, ViInt32* signal,
                     ViInt32* qualifier1, ViReal64* qualifier2);
```

LabVIEW Representation

```
AqDx Query Control IO Connectors.vi
```



Visual Basic Representation

```
GetControlIO (ByVal instrumentID As Long, _
              ByVal connector As Long, _
              signal As Long, _
              qualifier1 As Long, _
              qualifier2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getControlIO (ByVal instrumentID As Int32, _
              ByVal connector As Int32, _
              ByRef signal As Int32, _
              ByRef qualifier1 As Int32, _
              ByRef qualifier2 As Double) As Int32
```

2.3.34 AcqrsD1_getExtClock

Purpose

Returns the current external clock control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
clockType	ViInt32	= 0 Internal Clock (default at start-up) = 1 External Clock, continuously running = 2 External Reference (10 MHz) = 4 External Clock, with start/stop sequence
inputThreshold	ViReal64	Input threshold for external clock or reference in mV
delayNbrSamples	ViInt32	Number of samples to acquire after trigger (for 'clockType' = 1 only!)
inputFrequency	ViReal64	The presumed input frequency of the external clock, for clockType = 4 only
sampFrequency	ViReal64	The desired Sampling Frequency, for clockType = 4 only

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

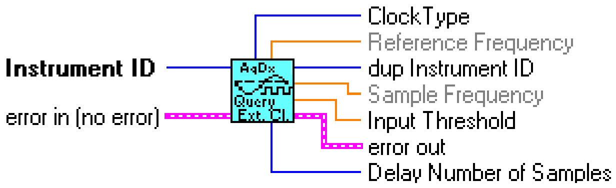
See remarks under **AcqrsD1_configExtClock**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getExtClock(ViSession instrumentID,
                      ViInt32* clockType, ViReal64* inputThreshold,
                      ViInt32* delayNbrSamples, ViReal64*
                      inputFrequency, ViReal64* sampFrequency);
```

LabVIEW Representation

AqDx Query External Clock.vi



Visual Basic Representation

```
GetExtClock (ByVal instrumentID As Long, _
             clockType As Long, _
             inputThreshold As Double, _
             delayNbrSamples As Long, _
             inputFrequency As Double, _
             sampFrequency As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getExtClock (ByVal instrumentID As Int32, _
             ByRef clockType As Int32, _
             ByRef inputThreshold As Double, _
             ByRef delayNbrSamples As Int32, _
             ByRef inputFrequency As Double, _
             ByRef sampFrequency As Double) As Int32
```

2.3.35 AcqrsD1_getFCounter

Purpose

Returns the current frequency counter configuration

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
signalChannel	ViInt32	Signal input channel
type	ViInt32	Type of measurement = 0 Frequency (default) = 1 Period (1/frequency) = 2 Totalize by Time = 3 Totalize by Gate
targetValue	ViReal64	User-supplied estimate of the expected value
apertureTime	ViReal64	Time in sec, during which the measurement is executed
reserved	ViReal64	Currently ignored
flags	ViInt32	Currently ignored

Return Value

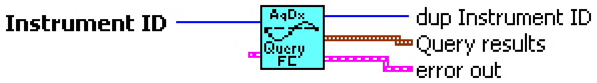
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getFCounter(ViSession instrumentID,
          ViInt32* signalChannel, ViInt32* type, ViReal64* targetValue,
          ViReal64* apertureTime, ViReal64* reserved, ViInt32* flags);
```

LabVIEW Representation

AqDx Query FCounter.vi



Visual Basic Representation

```
GetFCounter (ByVal instrumentID As Long, _
             signalChannel As Long, _
             type As Long, _
             targetValue As Double, _
             apertureTime As Double, _
             reserved As Double, _
             flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getFCounter (ByVal instrumentID As Int32, _
             ByRef signalChannel As Int32, _
             ByRef type As Int32, _
             ByRef targetValue As Double, _
             ByRef apertureTime As Double, _
             ByRef reserved As Double, _
             ByRef flags As Int32) As Int32
```

2.3.36 AcqrsD1_getHorizontal

Purpose

Returns the current horizontal control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
sampInterval	ViReal64	Sampling interval in seconds
delayTime	ViReal64	Trigger delay time in seconds

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

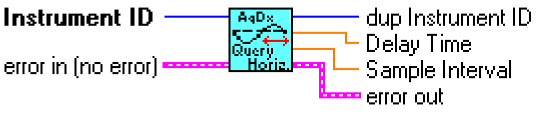
See remarks under **AcqrsD1_configHorizontal**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getHorizontal(ViSession instrumentID,
                     ViReal64* sampInterval, ViReal64* delayTime);
```

LabVIEW Representation

```
AqDx Query Horizontal Settings.vi
```



Visual Basic Representation

```
GetHorizontal (ByVal instrumentID As Long, _
               sampInterval As Double, _
               delayTime As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getHorizontal (ByVal instrumentID As Int32, _
               ByRef sampInterval As Double, _
               ByRef delayTime As Double) As Int32
```

2.3.37 AcqrsD1_getInstrumentData

Purpose

Returns some basic data about a specified digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
name	ViChar [ ]	Pointer to user-allocated string, into which the model name is returned (length < 32 characters).
serialNbr	ViInt32	Serial number of the digitizer.
busNbr	ViInt32	Bus number of the digitizer location.
slotNbr	ViInt32	Slot number of the digitizer location. (logical)

Return Value

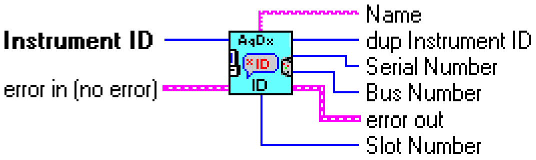
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getInstrumentData(ViSession instrumentID,
                    ViChar name[], ViInt32* serialNbr,
                    ViInt32* busNbr, ViInt32* slotNbr);
```

LabVIEW Representation

AqDx Query Instrument ID.vi



Visual Basic Representation

```
GetInstrumentData (ByVal instrumentID As Long, ByVal name As String, _
                  serialNbr As Long, busNbr As Long, _
                  slotNbr As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getInstrumentData (ByVal instrumentID As Int32, _
                  ByVal name As String, _
                  ByRef serialNbr As Int32, _
                  ByRef busNbr As Int32, _
                  ByRef slotNbr As Int32) As Int32
```

2.3.38 AcqrsD1_getInstrumentInfo

Purpose

Returns general information about a specified digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
parameterString	ViString	Character string defining the requested parameter. See below for the list of accepted strings.

Output

Name	Type	Description
infoValue	ViAddr	Requested information value. `ViAddr` resolves to `void*` in C/C++. The user must allocate the appropriate variable type (as listed below) and supply its address as 'infoValue'.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Accepted Parameter Strings

Parameter String	Returned Type	Description
"ASBus_*m*_BusNb"	ViInt32	Bus number of the *m*'th module of a multi-instrument. *m* runs from 0 to (nbr of modules –1).
"ASBus_ *m*_IsMaster"	ViInt32	Physical crate number of the *m*'th module of a multi-instrument (perhaps from AqGeo.map). *m* runs from 0 to (nbr of modules –1).
"ASBus_ *m*_PosInCrate"	ViInt32	Physical slot number (position) in cPCI crate of the *m* 'th module of a multi-instrument. *m* runs from 0 to (nbr of modules –1).
"ASBus_ *m*_SerialNb"	ViInt32	Serial number of the *m*'th module of a multi-instrument. *m* runs from 0 to (nbr of modules –1).
"ASBus_ *m*_SlotNb"	ViInt32	Slot number of the *m*'th module of a multi-instrument. *m* runs from 0 to (nbr of modules –1).
"CrateNb"	ViInt32	Physical crate number (perhaps from AqGeo.map)
"DelayOffset"	ViReal64	Calibrated Delay Offset (only useful for recovery of battery backed-up acquisitions)
"DelayScale"	ViReal64	Calibrated Delay Scale (only useful for recovery of battery backed-up acquisitions)
"ExtCkRatio"	ViReal64	Ratio of sFmax over external clock inputFrequency
"HasTrigVeto"	ViInt32	Returns 1 if the functionality is available, 0 otherwise.
"IsPreTriggerRunning"	ViInt32	Returns 1 if the module has an acquisition started but is not yet ready to accept a trigger.
"LOGDEVHDRBLOCK*m*DEV*n*S *string*"	ViChar[ ]	Returns information about FPGA firmware loaded. See comments below.
"MaxSamplesPerChannel"	ViInt32	Max. Number of samples per channel available in digitizer mode
"NbrADCBits"	ViInt32	Number of bits of data per sample from this modules ADCs
"NbrExternalTriggers"	ViInt32	Number of external trigger sources
"NbrInternalTriggers"	ViInt32	Number of internal trigger sources
"NbrModulesInInstrument"	ViInt32	Number of modules in this instrument. Individual modules

Parameter String	Returned Type	Description
		(not connected through ASBus) return 1.
`"Options"`	ViChar[ ]	List of options, separated by ',', installed in this instrument.
`"OverloadStatus chan"`	ViInt32	Returns 1 if *chan* is in overload, 0 otherwise. *chan* takes on the same values as 'channel' in **AcqrsD1_configTrigSource**.
`"OverloadStatus ALL"`	ViInt32	Returns 1 if any of the signal or external trigger inputs is in overload, 0 otherwise. Use the "OverloadStatus *chan* " string to determine which channel is in overload.
`"PosInCrate"`	ViInt32	Physical slot number (position) in cPCI crate
`"TbSegmentPad"`	ViInt32	Returns the additional array space (in samples) per segment needed for the image read of **AcqrsD1_readData**, **AcqrsD1_readCharSequence** or **AcqrsD1_readRealSequence** (DEPRECATED).
`"Temperature m"`	ViInt32	Temperature in degrees Centigrade (°C)
`"TrigLevelRange chan"`	ViReal64	Trigger Level Range on channel *chan*
`“VersionUserDriver”`	ViChar[ ]	String containing the full driver version.

Discussion

For the case "TrigLevelRange *chan*" the result is to be interpreted as ± (returned value), which is in % of the vertical Full Scale of the channel, or in mV for an external trigger source. The value of *chan* takes is the same as the values of 'channel' in **AcqrsD1_configTrigSource**.

For the case "Temperature *m*", *m* is the module number in a *MultiInstrument* and runs from 0 to (nbr of modules –1) following the channel order. It may be omitted on single digitizers or for the master of a *MultiInstrument*

For the case "Options" the available options are returned in a ',' separated string. The options include the memory size if additional memory has been installed in the form "MnM" for digitizers where n is the number of megabytes available or "PnMB" for AP235/AP240 and "AnM" for AP100/AP101/AP200/AP201. Other possible options include "NoASBus", "BtBkup", "FreqCntr", "SSR", "Avg", and "StrtOnTrig". The infoValue should point to a string of at least 32 characters.

The case of "LOGDEVHDRBLOCK*m*DEV*n*S *string*" is one in which several possible values of *m, n,* and *string* are allowed. The single digit number *m* refers to the FPGA block in the module. For the moment this must always have the value 1. The single digit number *n* refers to the FPGA device in the block. It can have values in the range 1,2,3,4 depending on the module. Among the interesting values of *string* are the following case-sensitive strings: "name", "version", "versionTxt", "compDate", "model".

Examples

```
double trigLevelRange;
AcqrsD1_getInstrumentInfo(ID, "TrigLevelRange -1",
                          &trigLevelRange);
```

The acceptable trigger levels are in the range [-*trigLevelRange*, +*trigLevelRange*] mV (external trigger!).

For modules supporting switch on overload protection:

```
long overLoad;
AcqrsD1_getInstrumentInfo(ID, "OverLoadStatus ALL", &overLoad);
if (overLoad)
     DO SOMETHING
```

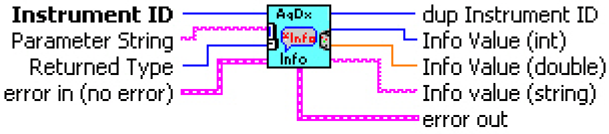
In order to find out which channel(s) caused the overload, you have to loop over `"OverLoadStatus -1", "OverLoadStatus 1", "OverLoadStatus 2",...`

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getInstrumentInfo(ViSession instrumentID,
                     ViString parameterString, ViAddr infoValue);
```

LabVIEW Representation

```
AqDx Query Instrument Information.vi
```



NOTE: The type of the returned value depends on the parameter requested. In LabVIEW, the correct returned type should be supplied as input to the VI, and the appropriate output wire connected. Any other wire will always return zero.

Visual Basic Representation

NOTE: In Visual Basic, a returned type of `ViInt32` should be declared as `Long`, while a returned type of `ViReal64` should be declared as `Double`.

```
GetInstrumentInfo (ByVal instrumentID As Long, _
                   ByVal parameterString As String, _
                   infoValue As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getInstrumentInfo (ByVal instrumentID As Int32, _
                   ByVal parameterString As String, _
                   ByRef infoValue As Int32) As Int32
or

AcqrsD1_getInstrumentInfo (ByVal instrumentID As Int32, _
                   ByVal parameterString As String, _
                   ByRef infoValue As Double) As Int32
or

AcqrsD1_getInstrumentInfo (ByVal instrumentID As Int32, _
                   ByVal parameterString As String, _
                   ByVal infoValue As String) As Int32
```

2.3.39 AcqrsD1_getMemory

Purpose

Returns the current memory control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nbrSamples	ViInt32	Nominal number of samples to record (per segment!)
nbrSegments	ViInt32	Number of segments to acquire. 1 corresponds to the normal single-trace acquisition mode.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

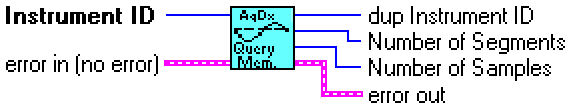
See remarks under **AcqrsD1_configMemory**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getMemory(ViSession instrumentID,
                     ViInt32* nbrSamples, ViInt32* nbrSegments);
```

LabVIEW Representation

```
AqDx Query Memory Settings.vi
```



Visual Basic Representation

```
GetMemory (ByVal instrumentID As Long, _
           nbrSamples As Long, _
           nbrSegments As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getMemory (ByVal instrumentID As Int32, _
           ByRef nbrSamples As Int32, _
           ByRef nbrSegments As Int32) As Int32
```

2.3.40 AcqrsD1_getMode

Purpose

Returns the current operational mode of the digitizer

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
mode	ViInt32	Operational mode
modifier	ViInt32	Modifier, currently not used
flags	ViInt32	Flags

Return Value

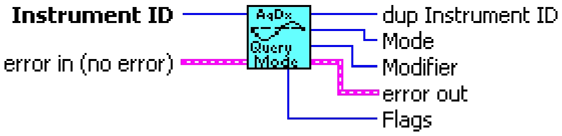
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getMode(ViSession instrumentID,
                   ViInt32* mode, ViInt32* modifier, ViInt32* flags);
```

LabVIEW Representation

```
AqDx Query Operation Mode.vi
```



Visual Basic Representation

```
GetMode (ByVal instrumentID As Long, _
         mode as Long, _
         modifier As Long, _
         flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getMode (ByVal instrumentID As Int32, _
         ByRef mode as Int32, _
         ByRef modifier As Int32, _
         ByRef flags As Int32) As Int32
```

2.3.41 AcqrsD1_getMultiInput

Purpose

Returns the multiple input configuration on a channel.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan

Output

Name	Type	Description
input	ViInt32	= 0 input connection A = 1 input connection B

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

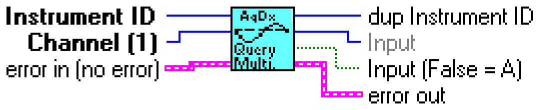
This function is only of use for instruments with an input-multiplexer (i.e. more than 1 input per digitizer, e.g. DP211). On the "normal" instruments with a single input per channel, this function may be ignored.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getMultiInput(ViSession instrumentID,
                   ViInt32 channel, ViInt32* input);
```

LabVIEW Representation

```
AqDx Query Multiplexer Input.vi
```



Visual Basic Representation

```
GetMultiInput (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               inputs As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getMultiInput (ByVal instrumentID As Int32, _
               ByVal channel As Int32, _
               ByRef input As Int32) As Int32
```

2.3.42 AcqrsD1_getNbrChannels

Purpose

Returns the number of channels on the specified module.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nbrChannels	ViInt32	Number of channels in the specified module

Return Value

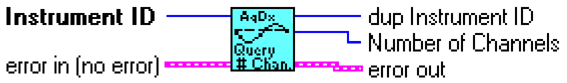
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getNbrChannels(ViSession instrumentID,
                     ViInt32* nbrChannels);
```

LabVIEW Representation

```
AqDx Query Number of Channels.vi
```



Visual Basic Representation

```
GetNbrChannels (ByVal instrumentID As Long, _
                nbrChannels As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getNbrChannels (ByVal instrumentID As Int32, _
                ByRef nbrChannels As Int32) As Int32
```

2.3.43 AcqrsD1_getNbrPhysicalInstruments

Purpose

Returns the number of physical Acqiris modules found on the computer.

Parameters

Output

Name	Type	Description
nbrInstruments	ViInt32	Number of Acqiris modules found on the computer

Return Value

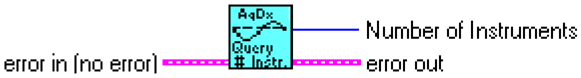
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getNbrPhysicalInstruments(
                     ViInt32* nbrInstruments);
```

LabVIEW Representation

```
AqDx Query Number of Instruments.vi
```



Visual Basic Representation

```
GetNbrPhysicalInstruments (nbrInstruments As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getNbrPhysicalInstruments (ByRef nbrInstruments As Int32 _
         ) As Int32
```

2.3.44 AcqrsD1_getSetupArray

Purpose

Returns an array of configuration parameters.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
setupType	ViInt32	Type of setup. 0 = GateParameters
nbrSetupObj	ViInt32	Maximum allowed number of setup objects in the output.

Output

Name	Type	Description
setupData	ViAddr	Pointer to an array for the setup objects `ViAddr` resolves to `void*` in C/C++. The user must allocate the appropriate array and supply its address as 'setupData'
nbrSetupObj-Returned	ViAddr	Number of setup objects returned

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

AqGateParameters

Name	Type	Description
GatePos	ViInt32	Start position of the gate
GateLength	ViInt32	Length of the gate

Discussion

For the object definition refer to **AcqrsD1_configSetupArray**. If **AcqrsD1_getSetupArray** is called without having set the Parameters before, the default values will be returned.

Note: The driver contains a set of 64 default AqGateParameters, defined as { {0,256} {256, 256} {512, 256} {768, 256} ... }.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getSetupArray(ViSession instrumentID,
                     ViInt32 channel,
                     ViInt32 setupType, ViInt32 nbrSetupObj
                     ViAddr setupData, ViInt32 nbrSetupObjReturned);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
GetSetupArray (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               ByVal setupType As Long, _
               ByVal nbrSetupObj As Long, _
               setupData As Any, _
               nbrSetupObjReturned As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getSetupArray (ByVal instrumentID As Int32, _
               ByVal channel As Int32, _
               ByVal setupType As Int32, _
               ByVal nbrSetupObj As Int32, _
               ByRef setupData As Int32, _
               ByRef nbrSetupObjReturned As Int32) As Int32
```

2.3.45 AcqrsD1_getTrigClass

Purpose

Returns the current trigger class control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
trigClass	ViInt32	= 0 edge trigger = 1 TV trigger
sourcePattern	ViInt32	= 0x000n0001 for Channel 1, = 0x000n0002 for Channel 2, = 0x000n0004 for Channel 3, = 0x000n0008 for Channel 4 etc. = 0x800n0000 for External Trigger 1, = 0x400n0000 for External Trigger 2 etc. where n is 0 for single instruments, or the module number for *MultiInstruments* (ASBus operation). See discussion below.
validatePattern	ViInt32	Currently returns "0"
holdType	ViInt32	Currently returns "0"
holdValue1	ViReal64	Currently returns "0"
holdValue2	ViReal64	Currently returns "0"

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

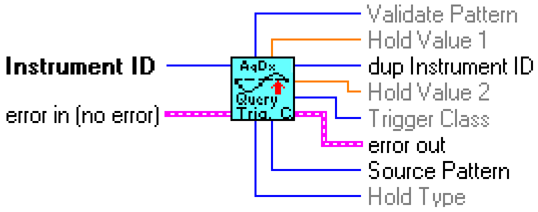
See remarks under **AcqrsD1_configTrigClass**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getTrigClass(ViSession instrumentID,
                      ViInt32* trigClass, ViInt32* sourcePattern,
                      ViInt32* validatePattern, ViInt32* holdType,
                      ViReal64* holdValue1, ViReal64* holdValue2);
```

LabVIEW Representation

AqDx Query Trigger Class.vi



Visual Basic Representation

```
GetTrigClass (ByVal instrumentID As Long, _
              trigClass As Long, _
              sourcePattern As Long, _
              validatePattern As Long, _
              holdType As Long, _
              holdValue1 As Double, _
              holdValue2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getTrigClass (ByVal instrumentID As Int32, _
              ByRef trigClass As Int32, _
              ByRef sourcePattern As Int32, _
              ByRef validatePattern As Int32, _
              ByRef holdType As Int32, _
              ByRef holdValue1 As Double, _
              ByRef holdValue2 As Double) As Int32
```

2.3.46 AcqrsD1_getTrigSource

Purpose

Returns the current trigger source control parameters for a specified channel.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	= 1...(Number of IntTrigSources) for internal sources = -1..-(Number of ExtTrigSources) for external sources See discussion below.

Output

Name	Type	Description
trigCoupling	ViInt32	= 0 DC = 1 AC = 2 HF Reject = 3 DC, 50 Ω = 4 AC, 50 Ω
trigSlope	ViInt32	= 0 Positive = 1 Negative = 2 out of Window = 3 into Window = 4 HF divide
trigLevel1	ViReal64	Trigger threshold in % of the vertical Full Scale of the channel, or in mV if using an External trigger source. See discussion below.
trigLevel2	ViReal64	Trigger threshold 2 (as above) for use when Window trigger is selected

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

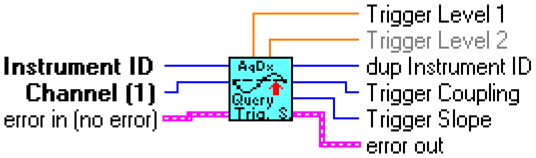
See remarks under **AcqrsD1_configTrigSource**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getTrigSource(ViSession instrumentID,
                     ViInt32 channel, ViInt32* trigCoupling,
                     ViInt32* trigSlope, ViReal64* trigLevel1,
                     ViReal64* trigLevel2);
```

LabVIEW Representation

```
AqDx Query Trigger Source.vi
```



Visual Basic Representation

```
GetTrigSource (ByVal instrumentID As Long, _
               ByVal Channel As Long, _
               trigCoupling As Long, _
               trigSlope As Long, _
               trigLevel1 As Double, _
               trigLevel2 As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getTrigSource (ByVal instrumentID As Int32, _
               ByVal Channel As Int32, _
               ByRef trigCoupling As Int32, _
               ByRef trigSlope As Int32, _
               ByRef trigLevel1 As Double, _
               ByRef trigLevel2 As Double) As Int32
```

2.3.47 AcqrsD1_getTrigTV

Purpose

Returns the current TV trigger parameters (12-bit-FAMILY only).

Parameters

Input

Name	Type	Description
InstrumentID	ViSession	Instrument identifier
Channel	ViInt32	= -1..-(Number of ExtTrigSources) for external sources See discussion below.

Output

Name	Type	Description
Standard	ViInt32	= 0 625/50/2:1 (PAL or SECAM) = 2 525/60/2:1 (NTSC)
Field	ViInt32	= 1 Field 1 - odd = 2 Field 2 - even
Line	ViInt32	= line number, depends on the parameters above: For 'standard' = 625/50/2:1 = 1 to 313 for 'field' = 1 = 314 to 625 for 'field' = 2 For 'standard' = 525/60/2:1 = 1 to 263 for 'field' = 1 = 1 to 262 for 'field' = 2

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

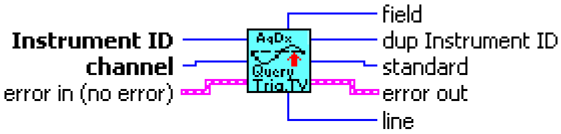
See discussion under **AcqrsD1_configTrigTV**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getTrigTV (ViSession instrumentID, ViInt32
                     channel, ViInt32* standard,
                     ViInt32* field, ViInt32* line);
```

LabVIEW Representation

AqDx Query Trigger TV.vi



Visual Basic Representation

```
GetTrigTV (ByVal instrumentID As Long, _
                 ByVal Channel As Long, _
                 standard As Long, _
                 field As Long, _
                 line AS Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getTrigTV (ByVal instrumentID As Int32, _
                 ByVal Channel As Int32, _
                 ByRef standard As Int32, _
                 ByRef field As Int32, _
                 ByRef line AS Int32) As Int32
```

2.3.48 AcqrsD1_getVersion

Purpose

Returns version numbers associated with a specified digitizer or current device driver.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
versionItem	ViInt32	1 for version of Kernel-Mode Driver 2 for version of EEPROM Common Section 3 for version of EEPROM Digitizer Section 4 for version of CPLD firmware

Output

Name	Type	Description
version	ViInt32	version number of the requested item

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

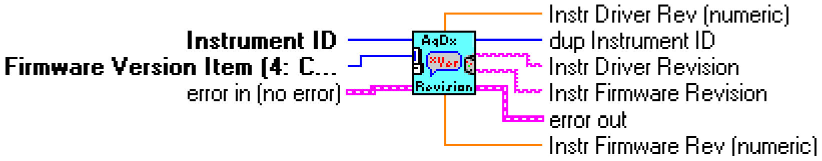
For drivers, the version number is composed of 2 parts. The upper 2 bytes represent the major version number, and the lower 2 bytes represent the minor version number.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getVersion(ViSession instrumentID,
                        ViInt32 versionItem, ViInt32* version);
```

LabVIEW Representation

```
AqDx Revision Query.vi
```



Visual Basic Representation

```
GetVersion (ByVal instrumentID As Long, _
            ByVal versionItem As Long, version As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getVersion (ByVal instrumentID As Int32, _
        ByVal versionItem As Int32, ByRef version As Int32) As Int32
```

2.3.49 AcqrsD1_getVertical

Purpose

Returns the vertical control parameters for a specified channel in the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan, or –1,… for the External Input

Output

Name	Type	Description
fullScale	ViReal64	in Volts
offset	ViReal64	in Volts
coupling	ViInt32	= 1 DC, 1 MΩ = 2 AC, 1 MΩ = 3 DC, 50 Ω = 4 AC, 50 Ω
bandwidth	ViInt32	= 0 no bandwidth limit (default) = 1 bandwidth limit at 25 MHz = 2 bandwidth limit at 700 MHz = 3 bandwidth limit at 200 MHz = 4 bandwidth limit at 20 MHz = 5 bandwidth limit at 35 MHz

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

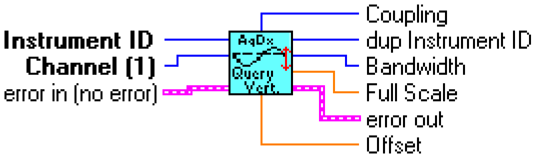
See remarks under **AcqrsD1_configVertical**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getVertical(ViSession instrumentID,
                     ViInt32 channel, ViReal64* fullScale,
                     ViReal64* offset, ViInt32* coupling,
                     ViInt32* bandwidth);
```

LabVIEW Representation

```
AqDx Query Vertical Settings.vi
```



Visual Basic Representation

```
GetVertical (ByVal instrumentID As Long, _
             ByVal Channel As Long, _
             fullScale As Double, _
             offset As Double, _
             coupling As Long, _
             bandwidth As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getVertical (ByVal instrumentID As Int32, _
             ByVal Channel As Int32, _
             ByRef fullScale As Double, _
             ByRef offset As Double, _
             ByRef coupling As Int32, _
             ByRef bandwidth As Int32) As Int32
```

2.3.50 AcqrsD1_init

Purpose

Initializes an instrument.

Parameters

Input

Name	Type	Description
resourceName	ViRsrc	ASCII string which identifies the digitizer to be initialized. See discussion below.
IDQuery	ViBoolean	Currently ignored
resetDevice	ViBoolean	If set to 'TRUE', resets the digitizer after initialization.

Output

Name	Type	Description
InstrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
Status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

You should refer to the **Programmer's Guide** section 3.2, **Device Initialization**, for a detailed explanation on the initialization procedure.

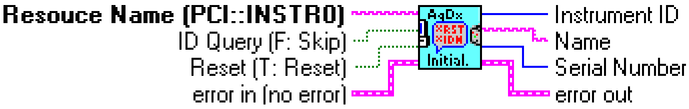
The function returns the error code ACQIRIS_ERROR_INIT_STRING_INVALID when the initialization string could not be interpreted.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_init(ViRsrc resourceName, ViBoolean IDQuery,
                     ViBoolean resetDevice, ViSession* instrumentID);
```

LabVIEW Representation

```
AqDx Initialize.vi
```



Visual Basic Representation

```
Init (ByVal resourceName As String, ByVal IDQuery As Boolean, _
    ByVal resetDevice As Boolean, instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_init (ByVal resourceName As String, ByVal IDQuery As Boolean,_
   ByVal resetDevice As Boolean, ByRef instrumentID As Int32) As Int32
```

2.3.51 AcqrsD1_InitWithOptions

Purpose

Initializes an instrument with options.

Parameters

Input

Name	Type	Description
resourceName	ViRsrc	ASCII string which identifies the digitizer to be initialized. See discussion below.
IDQuery	ViBoolean	Currently ignored
resetDevice	ViBoolean	If set to 'TRUE', resets the digitizer after initialization.
optionsString	ViString	ASCII string that specifies options. Syntax: "optionName=bool" where bool is TRUE (1) or FALSE (0). Currently three options are supported: "CAL": do calibration at initialization (default 1) "DMA": use scatter-gather DMA for data transfers (default 1). "simulate": initialize a simulated device (default 0). NOTE: **optionsString** is case insensitive.

Output

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

You should refer to the **Programmer's Guide** section 3.2, **Device Initialization** for a detailed explanation on the initialization procedure.

The function returns the error code ACQIRIS_ERROR_INIT_STRING_INVALID when the initialization string could not be interpreted.

When setting the option simulate to 1 (TRUE), the function **AcqrsD1_setSimulationOptions** should be called first with the appropriate options.

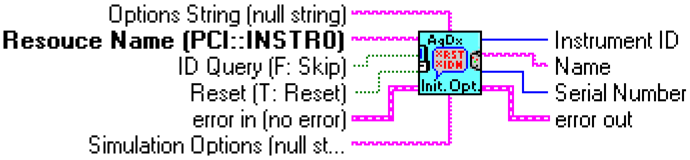
Multiple options can be given; Separate the option=value pairs with ',' characters.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_InitWithOptions(ViRsrc resourceName,
                        ViBoolean IDQuery, ViBoolean resetDevice,
                        ViString optionsString, ViSession* instrumentID);
```

LabVIEW Representation

```
AqDx Initialize with Options.vi
```



Visual Basic Representation

```
InitWithOptions (ByVal resourceName As String, _
                ByVal IDQuery As Boolean, _
                ByVal resetDevice As Boolean, _
                ByVal optionsString As String, _
                instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_InitWithOptions (ByVal resourceName As String, _
                ByVal IDQuery As Boolean, _
                ByVal resetDevice As Boolean, _
                ByVal optionsString As String, _
                ByRef instrumentID As Int32) As Int32
```

2.3.52 AcqrsD1_multiInstrAutoDefine

Purpose

Automatically initializes all digitizers and combines as many as possible to *MultiInstruments*. Digitizers are only combined if they are physically connected via ASBus.

Parameters

Input

Name	Type	Description
optionsString	ViString	ASCII string which specifies options. Currently no options are supported.

Output

Name	Type	Description
nbrInstruments	ViInt32	Number of user-accessible instruments. It also includes single instruments that don't participate on the ASBus.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This call must be followed by **nbrInstruments** calls to the functions **AcqrsD1_init** or **AcqrsD1_InitWithOptions** to retrieve the **instrumentID** of the (multi)digitizers.

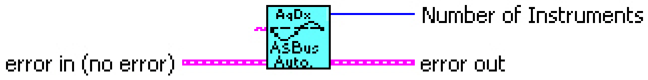
You should refer to to the **Programmer's Guide** section 3.2, **Device Initialization**, for a detailed explanation on the initialization procedure.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_multiInstrAutoDefine(ViString optionsString,
                     ViInt32* nbrInstruments);
```

LabVIEW Representation

```
AqDx MultiInstrument Auto Define.vi
```



Visual Basic Representation

```
MultiInstrAutoDefine (ByVal optionsString As String, _
                      nbrInstruments As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_multiInstrAutoDefine (ByVal optionsString As String, _
                      ByRef nbrInstruments As Int32) As Int32
```

2.3.53 AcqrsD1_multiInstrDefine

Purpose

This function defines the combination of a number of digitizers connected by ASBus into a single *MultiInstrument*.

Parameters

Input

Name	Type	Description
instrumentList	ViSession [ ]	Array of 'instrumentID' of already initialized single digitizers
nbrInstruments	ViInt32	Number of digitizers in the 'instrumentList'
masterID	ViSession	'instrumentID' of master digitizer

Output

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

You should refer to to the **Programmer's Guide** section 3.2, **Device Initialization,** for a detailed explanation on the initialization procedure.

The function returns the error code ACQIRIS_ERROR_MODULES_NOT_ON_SAME_BUS if all modules in the **instrumentList** are not on the same bus.

It may also return the error codes ACQIRIS_ERROR_NOT_ENOUGH_DEVICES or ACQIRIS_ERROR_NO_MASTER_DEVICE, when **nbrInstruments** is < 2 or the **masterID** is not one of the values in the **instrumentList**.

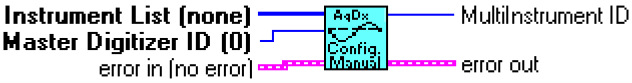
This function should only be used if the choices of the automatic initialization function **AcqrsD1_multiInstrAutoDefine** must be overridden. If the function executes successfully, the **instrumentID** presented in the **instrumentList** cannot be used anymore, since they represent individual digitizers that have become part of the new *MultiInstrument*, identified with newly returned **instrumentID**. Please refer to the **Programmer's Guide** section 3.2.6, **Manual Definition of MultiInstruments** for more information.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_multiInstrDefine(ViSession instrumentList[],
                        ViInt32 nbrInstruments, ViSession masterID,
                        ViSession* instrumentID);
```

LabView Representation

```
AqDx Configure MultiInstrument Manual Define.vi
```



Visual Basic Representation

```
MultiInstrDefine (ByRef instrumentList As Long, _
                  ByVal nbrInstruments As Long, _
                  ByVal masterID As Long, _
                  instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_multiInstrDefine (ByRef instrumentList As Int32, _
                  ByVal nbrInstruments As Int32, _
                  ByVal masterID As Int32, _
                  ByRef instrumentID As Int32) As Int32
```

2.3.54 AcqrsD1_multiInstrUndefineAll

Purpose

Undefines all *MultiInstruments*.

Parameters

Input

Name	Type	Description
optionsString	ViString	ASCII string which specifies options. Currently no options are supported.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

You should refer to to the **Programmer's Guide** section 3.2, **Device Initialization**, for a detailed explanation on the initialization procedure.

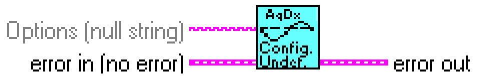
This function is almost never needed, except if you want to dynamically redefine *MultiInstruments* with the aid of the function **AcqrsD1_multiInstrDefine**. If the function executes successfully, the **instrumentID** of the previously defined *MultiInstruments* cannot be used anymore. You must either have remembered the **instrumentID** of the single instruments that made up the *MultiInstruments*, or you must reestablish all **instrumentID** of all digitizers by reinitializing with the code shown in the **Programmer's Guide** section 3.2.1, **Identification by Order Found**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_multiInstrUndefineAll(ViString
                     optionsString);
```

LabVIEW Representation

```
AqDx Configure MultiInstrument Undefine.vi
```



Visual Basic Representation

```
MultiInstrUndefineAll (ByVal optionsString As String) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_multiInstrUndefineAll (ByVal optionsString As String) As Long
```

2.3.55 AcqrsD1_procDone

Purpose

Checks if the on-board data processing has terminated. This routine is for Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
done	ViBoolean	done = VI_TRUE if the processing is terminated VI_FALSE otherwise

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_procDone(ViSession instrumentID,
                     ViBoolean* done);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
ProcDone (ByVal instrumentID As Long, done As Boolean) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_procDone (ByVal instrumentID As Int32, _
         ByRef done As Boolean) As Int32
```

2.3.56 AcqrsD1_processData

Purpose

Starts on-board data processing on acquired data in the current bank as soon as the current acquistion terminates. It can also be used to allow the following acquisition to be started as soon as possible. This routine is for Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
processType	ViInt32	Type of processing 0 = no processing 1 = gated peak detection, extrema mode 2 = gated peak detection, hysteresis mode 3 = interpolated peaks, extrema mode 4 = interpolated peaks, hysteresis mode
flags	ViInt32	Autoswitch functionality 0 = do (re-)processing in same bank 1 = start the next acquisition in the other bank 2 = switch banks but do not start next acquisition

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_processData(ViSession instrumentID,
                     ViInt32 processType, ViInt32 flags);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
ProcessData (ByVal instrumentID As Long, _
             ByVal processType As Long, _
             ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_processData (ByVal instrumentID As Int32, _
             ByVal processType As Int32, _
             ByVal flags As Int32) As Int32
```

2.3.57 AcqrsD1_readCharSequence (DEPRECATED)

Purpose

Returns a sequence of waveforms as a byte array.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
firstSegment	ViInt32	Requested first segment number, may assume 0 to the (number of segments – 1).
nbrSegments	ViInt32	Requested number of segments, may assume 1 to the number of segments set with the function **AcqrsD1_configMemory**.
firstSampleInSeg	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1), as set with the function **AcqrsD1_configMemory**.
nbrSamplesInSeg	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **AcqrsD1_configMemory**.
segmentOffset	ViInt32	Requested offset, in number of samples, between adjacent segments in the destination buffer *waveformArray*. Must be ≥ *nbrSamplesInSeg*.
arraySize	ViInt32	Number of data elements in the user-allocated *waveformArray*. Used for verification / protection.

Output

Name	Type	Description
waveformArray	ViChar [ ]	User-allocated waveform destination array of type char or byte. See discussion below for the required size.
horPos	ViReal64 [ ]	User-allocated array for horizontal positions of first data point, one per segment. See discussion below.
sampTime	ViReal64	Sampling interval in seconds
vGain	ViReal64	Vertical gain in Volts/LSB. See discussion below.
vOffset	ViReal64	Vertical offset in Volts. See discussion below.
timeStampLo timeStampHi	ViInt32 [ ] ViInt32 [ ]	User-allocated arrays for low and high parts of the 64-bit trigger timestamp. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function is faster than a loop over **AcqrsD1_readCharWform,** if many short segments (< 10'000 samples/segment) are read. See the **Programmer's Guide**, **Appendix A: Estimating Data Transfer Times** for timing details.

The waveform destination array **waveformArray** must not only allocate enough space to hold the requested data, but also some additional space. This function achieves a higher transfer speed by simply transferring an image of the digitizer memory to the CPU memory, and then reordering all circular segment buffers into linear arrays. Since allocating a temporary buffer for the memory image is time consuming, the user-allocated destination buffer is also used as a temporary storage for the memory image. The rule for the minimum storage space to allocate with **waveformArray** is discussed in to the **Programmer's Guide** section 3.8.2, **Reading Sequences of Waveforms**.

The value of *segmentOffset* must be ≥ *nbrSamplesInSeg*. The waveforms are thus transferred sequentially into a single linear buffer, with 'holes' of length (*segmentOffset – nbrSamplesInSeg*) between them. Such 'holes' could be used for depositing additional segment-specific information before storing the entire sequence as a single array to disk. If you specify *firstSegment* > 0, you don't have to allocate any buffer space for waveforms that are not read, i.e. **waveformArray[0]** corresponds to the first sample of the segment *firstSegment*.

Example: In a DC270, if you specify *nbrSamplesInSeg* = *segmentOffset* = 1500. Then with *nbrSegments* = 80 and *nbrSamplesNom* = 1000, since the *currentSegmentPad* = 408, you would have to allocate at least 1408 * (80 + 1) = 114'048 bytes.

It is strongly recommended to allocate the waveform destination buffers permanently rather than dynamically, in order to avoid system overheads for buffer allocation/deallocation.

The arrays **horPos**, **timeStampLo** and **timeStampHi** must always be allocated with length that corresponds to the total number of segments requested with the function **AcqrsD1_configMemory**. The timestamp of the first requested segment is therefore deposited in **timeStampLo[firstSegment]**, which is not necessarily **timeStampLo[0]**.

The returned parameters **horPos[ ]** are the horizontal positions, for each segment, of the first (nominal) data point with respect to the origin of the nominal trigger delay in seconds. Since the first data point is BEFORE the origin, this number will be in the range [-**sampTime**, 0]. Refer to the **Programmer's Guide** section 3.9, **Trigger Delay and Horizontal Waveform Position**, for a detailed discussion of the value **delayTime**.

The returned parameters **timeStampLo[ ]** and **timeStampHi[ ]** are respectively the low and high parts of the 64-bit trigger timestamp, on per segment, in units of picoseconds. The timestamp is the trigger time with respect to an arbitrary time origin (typically the start-time of the acquisition), which is intended for the computation of time differences between segments of a Sequence acquisition. Please refer to the **Programmer's Guide** section 3.11, **Sequence Timestamps**, for a detailed explanation.

The value in Volts of a data point **data** in the returned **waveformArray** can be computed with the formula:

```
V = vGain * data - vOffset
```

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_readCharSequence(ViSession instrumentID,
                     ViInt32 channel,     ViInt32 firstSegment,
                     ViInt32 nbrSegments, ViInt32 firstSampleInSeg,
                     ViInt32 nbrSamplesInSeg, ViInt32 segmentOffset,
                     ViInt32 arraySize,   ViChar waveformArray[],
                     ViReal64 horPos[],   ViReal64* sampTime,
                     ViReal64* vGain,     ViReal64* vOffset,
                     ViInt32 timeStampLo[],ViInt32 timeStampHi[]);
```

LabVIEW Representation

```
AqDx Read Sequence in ADC.vi should be considered as obsolete.
Please use AqDx Read Digitizer Data.vi instead
```

Visual Basic Representation

```
ReadCharSequence (ByVal instrumentID As Long, _
                  ByVal channel As Long, _
                  ByVal firstSegment As Long, _
                  ByVal nbrSegments As Long, _
                  ByVal firstSampleInSeg As Long, _
                  ByVal nbrSamplesInSeg As Long, _
                  ByVal segmentOffset As Long, _
                  ByVal arraySize As Long, _
                  waveformArray As Byte, _
                  horPos As Double, _
                  sampTime As Double, _
                  vGain As Double, _
                  vOffset As Double, _
                  timeStampLo As Long, _
                  timeStampHi As Long) As Long
```

2.3.58 AcqrsD1_readCharWform (DEPRECATED)

Purpose

Returns a waveform as a byte array.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
segmentNumber	ViInt32	Requested segment number, may assume 0 to the (number of segments – 1) set with the function **`AcqrsD1_configMemory`**.
firstSample	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1) set with the function **`AcqrsD1_configMemory`**.
nbrSamples	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **`AcqrsD1_configMemory`**.

Output

Name	Type	Description
waveformArray	ViChar [ ]	User-allocated waveform destination array of type char or byte. Its size MUST be at least (nbrSamples + 32), for reasons of data alignment.
returnedSamples	ViInt32	Number of data samples actually returned
addrFirstPoint	ViInt32	Offset of the first valid data point, that of the first sample, in the destination array. It should always be in the range [0...31].
horPos	ViReal64	Horizontal position of first data point. See discussion below.
sampTime	ViReal64	Sampling interval in seconds
vGain	ViReal64	Vertical gain in Volts/LSB. See discussion below.
vOffset	ViReal64	Vertical offset in Volts. See discussion below.
timeStampLo timeStampHi	ViInt32 ViInt32	Low and high part of the 64-bit trigger timestamp. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The returned parameter **horPos** is the horizontal position of the first (nominal) data point with respect to the origin of the nominal trigger delay in seconds. Since the first data point is BEFORE the origin, this number will be in the range [-**sampTime**, 0]. Refer to the **Programmer's Guide** section 3.9, **Trigger Delay and Horizontal Waveform Position** for a detailed discussion of the value **delayTime**.

The returned parameters **timeStampLo** and **timeStampHi** are respectively the low and high parts of the 64-bit trigger timestamp, in units of picoseconds. The timestamp is the trigger time with respect to an arbitrary time origin (typically the start-time of the acquisition), which is intended for the computation of time differences between segments of a Sequence acquisition. Please refer to the **Programmer's Guide** section 3.11, **Sequence Timestamps,** for a detailed explanation.

The value in Volts of a data point **data** in the returned **waveformArray** can be computed with the formula:

```
V = vGain * data - vOffset
```

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_readCharWform(ViSession instrumentID,
                     ViInt32 channel,     ViInt32 segmentNumber,
                     ViInt32 firstSample, ViInt32 nbrSamples,
                     ViChar waveformArray[], ViInt32* returnedSamples,
                     ViInt32* addrFirstPoint, ViReal64* horPos,
                     ViReal64* sampTime,  ViReal64* vGain,
                     ViReal64* vOffset,   ViInt32* timeStampLo,
                     ViInt32* timeStampHi);
```

LabVIEW Representation

```
AqDx Read Waveform in ADC.vi should be considered as obsolete.
Please use AqDx Read RAW Data.vi instead
```

Visual Basic Representation

```
ReadCharWform (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               ByVal segmentNumber As Long, _
               ByVal firstSample As Long, _
               ByVal nbrSamples As Long, _
               waveformArray As Byte, _
               returnedSamples As Long, _
               addrFirstPoint As Long, _
               horPos As Double, _
               sampTime As Double, _
               vGain As Double, vOffset As Double, _
               timeStampLo As Long, timeStampHi As Long) As Long
```

2.3.59 AcqrsD1_readData

Purpose

Returns all waveform information. The sample data is returned in an array whose type is specified in the **AqReadParameters** structure.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
readPar	AqReadParameters	Requested parameters for the acquired waveform.

Output

Name	Type	Description
dataArray	ViAddr	User-allocated waveform destination array. The array size restrictions are given below. `ViAddr` resolves to `void*` in C/C++.
dataDesc	AqDataDescriptor	Waveform descriptor structure, containing waveform information that is common to all segments.
segDescArray	ViAddr	Segment descriptor structure array, containing data that is specific for each segment. The size of the array is defined by *nbrSegments* and the type by *readMode.*If *readMode* =4 there are no segment descriptors.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Read Parameters in AqReadParameters

Name	Type	Description
dataType	ViInt32	Type representation of the waveform 0 = 8-bit (char) = 1 byte 1 = 16-bit (short) = 2 bytes 2 = 32-bit (long) = 4 bytes 3 = 64-bit (double) = 8 bytes
readMode	ViInt32	readout mode of the digitizer 0 = standard waveform (single segment only) 1 = image read for sequence waveform 2 = averaged waveform (from an averager ONLY) 3 = gated waveform (from an AP101/AP201 ONLY) 4 = peaks (from an AP101/AP201 ONLY) 5 = short averaged waveform (from an averager) 6 = shifted short averaged waveform (from an averager) 7 = SSR waveform from an Analyzer
firstSegment	ViInt32	Requested first segment number, may assume 0 to the (number of segments – 1).
nbrSegments	ViInt32	Requested number of segments, may assume 1 to the actual number of segments.
firstSampleInSeg	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the actual (number of samples – 1).
nbrSamplesInSeg	ViInt32	Requested number of samples, may assume 1 to the actual number of samples.

segmentOffset	ViInt32	ONLY used for readMode = 1 in DIGITIZERS: Requested offset, in number of samples, between adjacent segments in the destination buffer *dataArray*. Must be ≥ *nbrSamplesInSeg*
dataArraySize	ViInt32	Number of bytes in the user-allocated *dataArray*. Used for verification / protection.
segDescArraySize	ViInt32	Number of bytes in the user-allocated *segDescArray*. Used for verification / protection.
flags	ViInt32	ONLY used for DIGITIZERS 0 = First data point is before delayTime after Trigger 1 = First data point is at a fixed number of points with respect to the resynchronized trigger
reserved	ViInt32	Reserved for future use
reserved2	ViReal64	Reserved for future use
reserved3	ViReal64	Reserved for future use

Segment Descriptor for Normal Waveforms (readMode = 0,1,3) in AqSegmentDescriptor

Name	Type	Description
horPos	ViReal64	Horizontal position of first data point.
timeStampLo timeStampHi	ViUInt32 ViUInt32	Low and high part of the 64-bit trigger timestamp. See discussion below.

Segment Descriptor for Averaged Waveforms (readMode = 2,5) in AqSegmentDescriptorAvg

Name	Type	Description
horPos	ViReal64	Horizontal position of first data point.
timeStampLo timeStampHi	ViUInt32 ViUInt32	Low and high part of the 64-bit trigger timestamp. See discussion below.
actualTriggersInSeg	ViUInt32	Number of actual triggers acquired in this segment
avgOvfl	ViInt32	Acquisition overflow. See discussion below.
avgStatus	ViInt32	Average depth and status. See discussion below.
avgMax	ViInt32	Max value in the sequence. See discussion below.
reserved1	ViReal64	Reserved for future use

Data Descriptor in AqDataDescriptor

Name	Type	Description
returnedSamplesPerSeg	ViInt32	Total number of data samples actually returned. DataArray[indexFirstPoint]… DataArray[indexFirstPoint+ returnedSamplesPerSeg-1]
indexFirstPoint	ViInt32	Offset of the first valid data point, that of the first sample, in the destination array. It should always be in the range [0...31]. It is not an offset in bytes but rather and index in units of samples that may occupy more than one byte.
sampTime	ViReal64	Sampling interval in seconds
vGain	ViReal64	Vertical gain in Volts/LSB. See discussion below.
vOffset	ViReal64	Vertical offset in Volts. See discussion below.
returnedSegments	ViInt32	Number of segments
nbrAvgWforms	ViInt32	Number of averaged waveforms (nominal) in segment
actualTriggersInAcqLo actualTriggersInAcqHi	ViUInt32 ViUInt32	Low and high part of the 64-bit count of the number of triggers taken for the entire acquisition
actualDataSize	ViUInt32	Actual length in bytes used at dataArray
missingValue	ViReal64	Reserved for future use
reserved3	ViReal64	Reserved for future use

Discussion

All structures used in this function can be found in the header file **AcqirisDataTypes.h**.

The type of the **dataArray** is determined from the **AqReadParameters** struct entry **dataType**.

The following **dataType** and **readMode** combinations are supported:

	0 = standard	1 = image	2 = averaged	3 = gated	4 = peaks	5 = short averaged	6 = shifted short averaged	7 = SSR
0 = Int8	8	8	-	X	-	-	-	X
1 = Int16	12	12	-	-	-	X	X	-
2 = Int32	-	-	X	-	-	-	-	-
3 = Real64	X	X	X	-	X	X	X	-

In this table '8' means that the functionality is available for 8-bit Digitizers and AP units in the digitizer mode while '12' means that it is available for 12-bit Digitizers. It must be remembered that 12-bit digitizers generate 12 or 13-bit data which will be transferred as 2 bytes with the data shifted so that the MSB of the data becomes the MSB of the 16-bit word, thus preserving the sign information. The vGain value is therefore not the gain of the ADC in volts/LSB but rather the volts/LSB of the 16-bit word.

The value in Volts of any integer data point **data** in the returned **dataArray** for a digitizer can be computed with the formula:

```
V = vGain * data – vOffset
```

Except in the case of Analyzers, the data points for dataType = 3 are in Volts and no conversion is needed. For Analyzers the data points are in units of the LSB of the ADC and must be converted using the formula above.

The 3 "averaged" modes correspond to:

2 – 24-bit data read as such into either Int32 32-bit integers or converted into volts for Real64,

5 – 16-bit data read of the least significant 16 bits of the 24-bit sum. The result is presented in either an Int16 array or converted into volts for Real 64. The user is responsible for treating any potential overflows,

6 – 16-bit data read of the most significant 16 bits of the 24-bit sum. The result is presented in either an Int16 array or converted into volts for Real 64. The user is responsible for treating any potential overflows.

It should also be noted that the interpretation of averager results was discussed in the **Programmer's Guide** section 3.8.4, **Reading an Averaged Waveform from an Averager** and 3.8.5, **Reading a RT Add/Subtract Averaged Waveform from an Averager.**

If **readMode** is set to gated, the **nbrSamplesInSeg** is set to the sum of the gate lengths.

The rules for the allocation of memory for the **dataArray** are as follows:

- For digitizers (or other modules used as such)
  - with readMode = 0 and dataType = 0, the array size in bytes **must** be at least (nbrSamplesInSeg+32).
  - with readMode = 0 and dataType = 1, the array size in words **must** be at least (nbrSamplesInSeg+32).
  - with readMode = 0 and dataType = 3, the array size in bytes must be at least 40.
  - with readMode = 1 the waveform destination array **dataArray** must not only allocate enough space to hold the requested data, but also some additional space. This function achieves a higher transfer speed by simply transferring an image of the digitizer memory to the CPU memory, and then reordering all circular segment

buffers into linear arrays. Since allocating a temporary buffer for the memory image is time consuming, the user-allocated destination buffer is also used as a temporary storage for the memory image. The rule for the minimum storage space to allocate with **waveformArray** is discussed in the **Programmer's Guide** section 3.8.2, **Reading Sequences of Waveforms.**

- For averagers readMode = 2, 5 or 6 are allowed and the size **must** be at least nbrSamplesInSeg* nbrSegments * size_of_dataType

- For analyzers
  - with readMode = 0,1,2 its size **must** be at least nbrSamplesInSeg* nbrSegments
  - with readMode = 3 the array size must be at least the sum of all gate lengths.
  - with readMode = 4 the array size must be 4*sizeof(double) * number of gates
  - for all other cases, its size, in bytes, **must** be at least nbrSamplesInSeg* nbrSegments*size_of_dataType

  For configuring gate parameters see the **User Manual: Family of Analyzers**

The value of **returnedSamplesPerSeg** for **readMode** = 7 is not useable and therefore set to 0.

The segment descriptor array **segDesc[]** must always be allocated with a length that corresponds to the total number of segments requested with **nbrSegments** in **AqReadParameters**. The first requested segment is therefore deposited in **SegDesc[0]**. The segment descriptor array must also be allocated with the correct structure type that depends on the **readMode**.

The returned segment descriptor values **timeStampLo** and **timeStampHi** are respectively the low and high parts of the 64-bit trigger timestamp, in units of picoseconds. The timestamp is the trigger time with respect to an arbitrary time origin (typically the start-time of the acquisition), which is intended for the computation of time differences between segments of a Sequence acquisition. Please refer to the **Programmer's Guide** section 3.11, **Sequence Timestamps,** for a detailed explanation.

The returned segment descriptor value **horPos** is the horizontal position, for the segment, of the first (nominal) data point with respect to the origin of the nominal trigger delay in seconds. Since the first data point is BEFORE the origin, this number will be in the range [-**sampTime**, 0]. Refer to the **Programmer's Guide** section 3.9, **Trigger Delay and Horizontal Waveform Position**, for a detailed discussion of the value **delayTime**. For Averaged Waveforms, the value of **horPos** will always be 0.

avgOvfl, avgStatus and **avgMax** will apply to Signal Averagers only. The features that they support have not yet been implemented.

The value of *segmentOffset* must be ≥ *nbrSamplesInSeg*. The waveforms are thus transferred sequentially into a single linear buffer, with 'holes' of length (*segmentOffset – nbrSamplesInSeg*) between them. Such 'holes' could be used for depositing additional segment-specific information before storing the entire sequence as a single array to disk. If you specify *firstSegment* > 0, you don't have to allocate any buffer space for waveforms that are not read, i.e. **waveformArray[0]** corresponds to the first sample of the segment *firstSegment.*

Example: In a DC270, if you specify *nbrSamplesInSeg* = *segmentOffset* = 1500. Then with *nbrSegments* = 80 and *nbrSamplesNom* = 1000, since the *currentSegmentPad* = 408, you would have to allocate at least 1408 * (80 + 1) = 114'048 bytes.

It is strongly recommended to allocate the waveform destination buffers permanently rather than dynamically, in order to avoid system overheads for buffer allocation/deallocation.

LabWindowsCVI/Visual C++ Representation

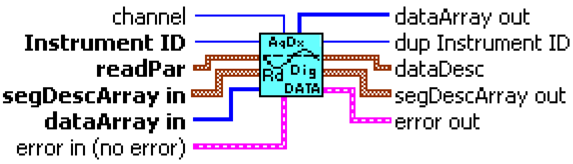
```
ViStatus status = AcqrsD1_readData(ViSession instrumentID,
                        ViInt32 channel, AqReadParameters* readPar,
                        ViAddr dataArray, AqDataDescriptor* descriptor,
                        ViAddr segDesc);
```

LabVIEW Representations

AqDx Read Digitizer Data.vi

This Vi is polymorphic, the sample data is returned in an array of type I8, I16 or DBL.

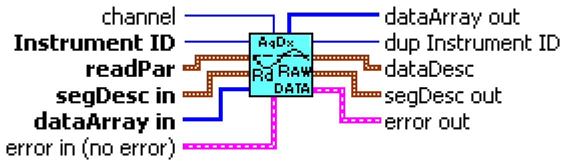
It is meant for the readout of multiple segments with readMode = 1.



AqDx Read Raw Data.vi

This Vi is polymorphic, the sample data is returned in an array of type I8, I16.

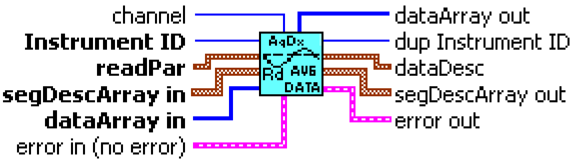
It is meant for the readout of a single segment with readMode = 0.



AqDx Read Averager Data.vi

This Vi is polymorphic, the sample data is returned in an array of type I32 or DBL

It is meant for the readout of an averager with readMode = 2.



Visual Basic Representation

```
ReadData (ByVal instrumentID As Long, _
          ByVal channel As Long, _
          readPar As AqReadParameters, _
          dataArray As Any, _
          dataDesc As AqDataDescriptor, _
          segDescArray As Any) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_readData (ByVal instrumentID As Int32, _
          ByVal channel As Int32, _
          ByRef readPar As AqReadParameters, _
          ByRef dataArray As DATATYPE, _
          ByRef dataDesc As AqDataDescriptor, _
          ByRef segDescArray As AqSegmentDescriptor) As Int32
Where DATATYPE can be either Int8, Int16, or Double

or

AcqrsD1_readData (ByVal instrumentID As Int32, _
          ByVal channel As Int32, _
          ByRef readPar As AqReadParameters, _
          ByRef dataArray As DATATYPEAVG, _
          ByRef dataDesc As AqDataDescriptor, _
          ByRef segDescArray As AqSegmentDescriptorAvg) As Int32 Int32
Where DATATYPEAVG can be either Int16, Int32, or Double
```

2.3.60 AcqrsD1_readFCounter

Purpose

Returns the result of a frequency counter measurement

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
result	ViReal64	Result of measurement

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The result must be interpreted as a function of the effected measurement ‘type’:

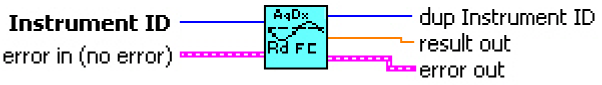
Measurement Type	Units
0 Frequency	Hz
1 Period	Sec
2 Totalize by Time	Counts
3 Totalize by Gate	Counts

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_readFCounter(ViSession instrumentID,
                      ViReal64* result);
```

LabVIEW Representation

```
AqDx Read FCounter.vi
```



Visual Basic Representation

```
ReadFCounter (ByVal instrumentID As Long, _
              result As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_readFCounter (ByVal instrumentID As Int32, _
              ByRef result As Double) As Int32
```

2.3.61 AcqrsD1_readRealSequence (DEPRECATED)

Purpose

Returns a sequence of waveforms as a floating point (double) array, with the measured data values in Volts.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
firstSegment	ViInt32	Requested first segment number, may assume 0 to the (number of segments – 1).
nbrSegments	ViInt32	Requested number of segments, may assume 1 to the number of segments set with the function **`AcqrsD1_configMemory`**.
firstSampleInSeg	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1), as set with the function **`AcqrsD1_configMemory`**.
nbrSamplesInSeg	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **`AcqrsD1_configMemory`**.
segmentOffset	ViInt32	Requested offset, in number of samples, between adjacent segments in the destination buffer *waveformArray*. Must be ≥ *nbrSamplesInSeg*.
arraySize	ViInt32	Number of data elements in the user-allocated *waveformArray*. Used for verification / protection.

Output

Name	Type	Description
waveformArray	ViReal64 [ ]	User-allocated waveform destination array of type double. See discussion below for the required size.
horPos	ViReal64 [ ]	User-allocated array for horizontal positions of first data point, one per segment. See discussion below.
sampTime	ViReal64	Sampling interval in seconds
timeStampLo timeStampHi	ViInt32 [ ] ViInt32 [ ]	User-allocated arrays for low and high parts of the 64-bit trigger timestamp. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

See remarks under **AcqrsD1_readCharSequence**for details about the **horPos** and **timeStamp** parameters and the **Programmer's Guide** section 3.8.2, **Reading Sequences of Waveforms**, for the allocation of the buffers. The **dataType** = 3 rule given there for the **arraySize** becomes

arraySize = *segmentOffset* * (*nbrSegments+1)*

since the **waveformArray** here is **ViReal64**. However, the other rule changes too

8*arraySize ≥ (*nbrSamplesNom* + *currentSegmentPad)* * (*nbrSegments+1)*

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_readRealSequence(ViSession instrumentID,
                        ViInt32 channel,      ViInt32 firstSegment,
                        ViInt32 nbrSegments,  ViInt32 firstSampleInSeg,
                        ViInt32 nbrSamplesInSeg, ViInt32 segmentOffset,
                        ViInt32 arraySize,    ViReal64 waveformArray[],
                        ViReal64 horPos[],    ViReal64* sampTime,
                        ViInt32 timeStampLo[],ViInt32 timeStampHi[]);
```

LabVIEW Representation

```
AqDx Read Sequence in Volts.vi should be considered as obsolete.
Please use AqDx Read Digitizer Data.vi instead
```

Visual Basic Representation

```
ReadRealSequence (ByVal instrumentID As Long, _
                  ByVal channel As Long, _
                  ByVal firstSegment As Long, _
                  ByVal nbrSegments As Long, _
                  ByVal firstSampleInSeg As Long, _
                  ByVal nbrSamplesInSeg As Long, _
                  ByVal segmentOffset As Long, _
                  ByVal arraySize As Long, _
                  waveformArray As Double, _
                  horPos As Double, _
                  sampTime As Double, _
                  timeStampLo As Long, _
                  timeStampHi As Long) As Long
```

2.3.62 AcqrsD1_readRealWform (DEPRECATED)

Purpose

Returns a waveform as a floating point (double) array, with the measured data values in Volts.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
segmentNumber	ViInt32	Requested segment number, may assume 0 to the (number of segments – 1) set with the function **`AcqrsD1_configMemory`**.
firstSample	ViInt32	Requested position of first sample to read, typically 0. May assume 0 to the (number of samples – 1) set with the function **`AcqrsD1_configMemory`**.
nbrSamples	ViInt32	Requested number of samples, may assume 1 to the number of samples set with the function **`AcqrsD1_configMemory`**.

Output

Name	Type	Description
waveformArray	ViReal64 [ ]	User-allocated waveform destination array. Its size MUST be at least the maximum of *nbrSamples or 5*.
returnedSamples	ViInt32	Number of data samples actually returned
horPos	ViReal64	Horizontal position of first data point. See discussion below.
sampTime	ViReal64	Sampling interval in seconds
timeStampLo timeStampHi	ViInt32 ViInt32	Low and high part of the 64-bit trigger timestamp. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

See remarks under **AcqrsD1_readCharWform** for details about the **horPos** and **timeStamp** parameters.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_readRealWform(ViSession instrumentID,
                     ViInt32 channel,     ViInt32 segmentNumber,
                     ViInt32 firstSample, ViInt32 nbrSamples,
                     ViReal64 waveformArray[], ViInt32* returnedSamples,
                     ViReal64* horPos,    ViReal64* sampTime,
                     ViInt32* timeStampLo, ViInt32* timeStampHi);
```

LabVIEW Representation

```
AqDx Read Waveform in Volts.vi should be considered as obsolete.
Please use AqDx Read Digitizer Data.vi instead
```

Visual Basic Representation

```
ReadRealWform (ByVal instrumentID As Long, _
               ByVal channel As Long, _
               ByVal segmentNumber As Long, _
               ByVal firstSample As Long, _
               ByVal nbrSamples As Long, _
               waveformArray As Double, _
               returnedSamples As Long, _
               horPos As Double, _
               sampTime As Double, _
               timeStampLo As Long, _
               timeStampHi As Long) As Long
```

2.3.63 AcqrsD1_reportNbrAcquiredSegments

Purpose

Returns the number of segments already acquired. For averagers (but not AP100 or AP200) it will give the number of triggers already accepted for the current acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nbrSegments	ViInt32	Number of segments already acquired

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

Can be called while an acquisition is active, in order to follow the progress of a Sequence and/or Averaging acquisition. As needed the result should be interpreted as a ViUInt32.

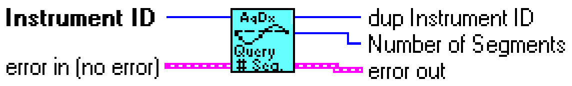
Can be called after an acquisition, in order to obtain the number of segments/triggers actually acquired (until **AcqrsD1_stopAcquisition** was called).

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_reportNbrAcquiredSegments(
                     ViSession instrumentID, ViInt32* nbrSegments);
```

LabVIEW Representation

```
AqDx Query Number of Acquired Segments.vi
```



Visual Basic Representation

```
ReportNbrAcquiredSegments (ByVal instrumentID As Long, _
                           nbrSegments As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_reportNbrAcquiredSegments (ByVal instrumentID As Int32, _
                ByRef nbrSegments As Int32) As Int32
```

2.3.64 AcqrsD1_reset

Purpose

Resets an instrument.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

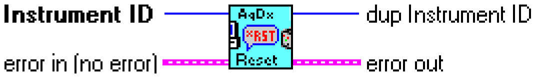
There is no known situation where this action is to be recommended.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_reset(ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Reset.vi
```



Visual Basic Representation

```
Reset (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_reset (ByVal instrumentID As Int32) As Int32
```

2.3.65 AcqrsD1_resetDigitizerMemory

Purpose

Resets the digitizer memory to a known default state.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

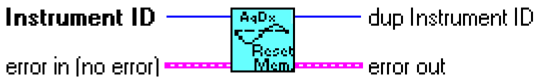
Each byte of the digitizer memory is overwritten sequentially with the values 0xaa, 0x55, 0x00 and 0xff. This functionality is mostly intended for use with battery backed-up memories.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_resetDigitizerMemory(
                        ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Reset Digitizer Memory.vi
```



Visual Basic Representation

```
ResetDigitizerMemory (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_resetDigitizerMemory (ByVal instrumentID As Int32) As Int32
```

2.3.66 AcqrsD1_restoreInternalRegisters

Purpose

Restores some internal registers of an instrument.
Only needed after power-up of a digitizer with the battery back-up option.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
delayOffset	ViReal64	Global delay offset, should be retrieved with **AcqrsD1_getInstrumentInfo**(…, "DelayOffset", ..) before power-off If not known, use the value –20.0e-9
delayScale	ViReal64	Global delay scale, should be retrieved with **AcqrsD1_getInstrumentInfo**(…, "DelayScale", ..) before power-off If not known, use the value 5.0e-12

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

The normal startup sequence destroys the contents of the Acqiris digitizer memories. This function, together with a specific sequence of other function calls, prevents this from occurring in digitizers with battery backed-up memories.

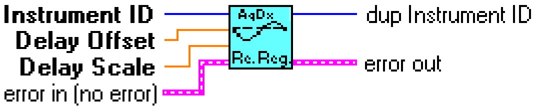
Please refer to the **Programmer's Guide** section 3.15, **Readout of Battery Backed-up Memories**, for a detailed description of the required initialization sequence to read battery backed-up waveforms.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_restoreInternalRegisters(
                      ViSession instrumentID, ViReal64 delayOffset,
                      ViReal64 delayScale);
```

LabVIEW Representation

```
AqDx Restore Internal Registers.vi
```



Visual Basic Representation

```
RestoreInternalRegisters (ByVal instrumentID As Long,
                          ByVal delayOffset As Double,
                          ByVal delayScale As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_restoreInternalRegisters (ByVal instrumentID As Int32,
                          ByVal delayOffset As Double,
                          ByVal delayScale As Double) As Int32
```

2.3.67 AcqrsD1_setLEDColor

Purpose

Sets the front panel LED to the desired color.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
color	ViInt32	0 = OFF (return to normal acquisition status indicator) 1 = Green 2 = Red 3 = Yellow

Return Value

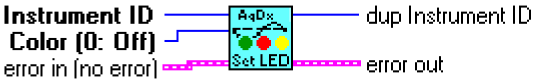
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_setLEDColor(ViSession instrumentID,
                    ViInt32 color);
```

LabVIEW Representation

```
AqDx Set LED Color.vi
```



Visual Basic Representation

```
SetLEDColor (ByVal instrumentID As Long, _
            ByVal color As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_setLEDColor (ByVal instrumentID As Int32, _
            ByVal color As Int32) As Int32
```

2.3.68 AcqrsD1_setSimulationOptions

Purpose

Sets one or several options which will be used by the function **AcqrsD1_InitWithOptions**, provided that the **optionsString** supplied to **AcqrsD1_InitWithOptions** contains the string "simulate=TRUE".

Parameters

Input

Name	Type	Description
simOptionString	ViString	String listing the desired simulation options. See discussion below.

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

See the **Programmer's Guide** section 3.2.8, **Simulated Devices,** for details on simulation. A string of the form "M8M" is used to set an 8 Mbyte simulated memory. The simulation options are reset to none by setting **simOptionString** to an empty string "".

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_setSimulationOptions(
                      ViString simOptionString);
```

LabVIEW Representation

```
Use AqDx Initialize with Options.vi
```

Visual Basic Representation

```
SetSimulationOptions (ByVal simOptionString As String) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_setSimulationOptions (ByVal simOptionString As String) _
                             As Int32
```

2.3.69 AcqrsD1_stopAcquisition

Purpose

Stops the acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

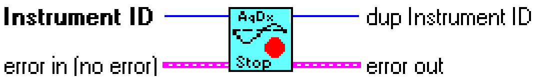
This function will stop the acquisition and not return until this has been accomplished. The data is not guaranteed to be valid. To obtain valid data after "manually" stopping the acquisition (e.g. timeout waiting for a trigger), one should use the **AcqrsD1_forceTrig** function to generate a "software" (or "manual") trigger, and then continue polling for the end of the acquisition with **AcqrsD1_acqDone**. This will ensure correct completion of the acquisition.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_stopAcquisition(ViSession instrumentID);
```

LabVIEW Representation

```
AqDx Stop Acquisition.vi
```



Visual Basic Representation

```
StopAcquisition (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_stopAcquisition (ByVal instrumentID As Int32) As Int32
```

2.3.70 AcqrsD1_stopProcessing

Purpose

Stops on-board data processing. This routine is for Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function will stop the on-board data processing immediately. The output data is not guaranteed to be valid.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_stopProcessing(ViSession instrumentID);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
StopProcessing (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_stopProcessing (ByVal instrumentID As Int32) As Int32
```

2.3.71 AcqrsD1_waitForEndOfAcquisition

Purpose

Waits for the end of acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
timeout	ViInt32	Timeout in milliseconds

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function will return only after the acquisition has terminated or when the requested timeout has elapsed, whichever is shorter. For protection, the timeout is clipped to a maximum value of 10 seconds. If a larger timeout is needed, call this function repeatedly.

While waiting for the acquisition to terminate, the calling thread is put into 'idle', permitting other threads or processes to fully use the CPU.

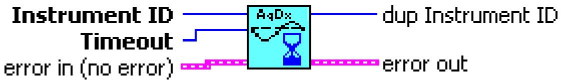
If a channel or trigger overload was detected, the returned status is always ACQIRIS_ERROR_OVERLOAD. Else, if the acquisition times out, the returned status is ACQIRIS_ERROR_ACQ_TIMEOUT, in which case you should use either **AcqrsD1_stopAcquisition** or **AcqrsD1_forceTrig** to stop the acquisition. Otherwise, the returned status is VI_SUCCESS.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_waitForEndOfAcquisition (ViSession
                     instrumentID, ViInt32 timeout);
```

LabVIEW Representation

```
AqDx Wait For End Of Acquisition.vi
```



Visual Basic Representation

```
WaitForEndOfAcquisition (ByVal instrumentID As Long, _
                         ByVal timeout As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_waitForEndOfAcquisition (ByVal instrumentID As Int32, _
                         ByVal timeout As Int32) As Int32
```

2.3.72 AcqrsD1_waitForEndOfProcessing

Purpose

Waits for the end of on-board data processing. . This routine is for Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
timeout	ViInt32	Timeout in milliseconds

Return Value

Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

Discussion

This function will return only after the on-board processing has terminated or when the requested timeout has elapsed, whichever is shorter. For protection, the timeout is clipped to a maximum value of 10 seconds. If a larger timeout is needed, call this function repeatedly.

While waiting for the processing to terminate, the calling thread is put into 'idle', permitting other threads or processes to fully use the CPU.

If the processing times out, the returned status is ACQIRIS_ERROR_PROC_TIMEOUT, in which case you should use **AcqrsD1_stopProcessing** to stop the processing. Otherwise, the returned status is VI_SUCCESS.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_waitForEndOfProcessing(ViSession
                     instrumentID, ViInt32 timeout);
```

LabVIEW Representation

Not yet supported

Visual Basic Representation

```
WaitForEndOfProcessing (ByVal instrumentID As Long, _
                        ByVal timeout As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_waitForEndOfProcessing (ByVal instrumentID As Int32, _
                        ByVal timeout As Int32) As Int32
```