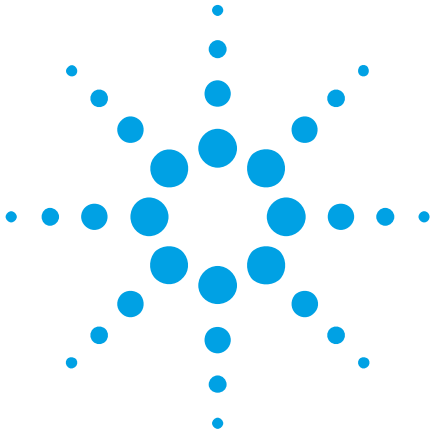


PROGRAMMER'S REFERENCE MANUAL AGILENT ACQIRIS INSTRUMENTS

Agilent Technologies

Manual Part Number

U1092-90002

Edition

I-RevA, February 2009

The information in this document is subject to change without notice and may not be construed in any way as a commitment by Agilent Technologies, Inc. While Agilent 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 Agilent or its affiliated companies.

You can download the latest version of this manual from http://www.agilent.com/ by clicking on Manuals in the Technical Support section and then entering a model number. You can also visit our web site at http://www.agilent.com/find/acqiris. At Agilent 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 Agilent Acqiris product line representative or the dedicated Agilent Acqiris Technical Support (ACQIRIS_SUPPORT@agilent.com).

Acqiris Product Line Information

USA (800) 829-4444

Asia - Pacific 61 3 9210 2890

Europe 41 (22) 884 32 90

© Copyright Agilent 2009

CONTENTS

1. Introduction

1.1. Message to the User

Congratulations on having purchased an Agilent Technologies Acqiris data conversion product. Acqiris Digitizers, Averagers, Analyzers, and Time-to-Digital Converters 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 2000/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, a digital oscilloscope, or other similar instrument.

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 3.3, an application that enables basic operation of Acqiris digitizers or averagers in a Windows 2000/XP environment;

the operation of SSR Demo and in the following chapter APx01 Demo, applications that enable basic operation of Acqiris analyzers in a Windows 2000/XP environment;

the operation of the demonstration program that enables basic operation of Acqiris Time-to-Digital Converters in a Windows 2000/XP environment;

the operation of Analyzer Demo, the demonstration program for the SC240/AC240/SC210/AC210 from a PC running a Windows 2000/XP operating system.

Chapter 5 ***RUNNING THE GEOMAPPER APPLICATION***, describes the purpose and operation of the GeoMapper application which is needed for some AS bus Multi-instrument systems.

The **Programmer's Guide** is divided into 3 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 INSTRUMENT***, provides information on using the device driver functions to operate an Acqiris instrument.

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, LabVIEW, MATLAB MEX, Visual Basic, Visual Basic .NET, Visual C++.

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/DC271AR/DP214/DP235/DP240
AP-FAMILY	AP240/AP235/AP100/AP101/AP200/AP201
12-bit-FAMILY	DC440/DC438/DC436/DP310/DP308/DP306
10-bit-FAMILY	DC122/DC152/DC222/DC252/DC282
U1071A-FAMILY	all U1071A variants, DP1400, U1091AD28

1.4. Warning Regarding Medical Use

The Agilent Acqiris 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 Instrument to which the call is directed. The only exceptions are the initialization/termination functions:

- **Acqrs_calibrate**
- **Acqrs_calibrateEx**
- **Acqrs_close**
- **Acqrs_closeAll**
- **Acqrs_getNbrInstruments**
- **Acqrs_init**
- **Acqrs_InitWithOptions**
- **Acqrs_setSimulationOptions**
- **AcqrsD1_close**
- **AcqrsD1_init**
- **AcqrsD1_InitWithOptions**
- **AcqrsD1_getNbrPhysicalInstruments**
- **AcqrsD1_multiInstrAutoDefine**
- **AcqrsD1_setSimulationOptions**
- **AcqrsD1_multiInstrUndefineAll**

The functions **Acqrs_init, Acqrs_InitWithOptions, 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_BUFFER_NOT_64BITS_ALIGNED	BFFA480A	-1074116598
ACQIRIS_ERROR_BUFFER_NOT_32BITS_ALIGNED	BFFA480B	-1074116597
ACQIRIS_ERROR_CAL_FILE_CORRUPTED	BFFA480C	-1074116596
ACQIRIS_ERROR_CAL_FILE_VERSION	BFFA480D	-1074116595
ACQIRIS_ERROR_CAL_FILE_SERIAL	BFFA480E	-1074116594
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_IO_DEVICE_OFF	BFFA48A2	-1074116446
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_EEPROM2_DATA_INVALID	BFFA48D2	-1074116398
ACQIRIS_ERROR_INSTRUMENT_IN_USE	BFFA48D3	-1074116397
ACQIRIS_ERROR_MEZZIO_IN_USE	BFFA48D4	-1074116396

Acqiris Error Codes	Hex value	Decimal value
ACQIRIS_ERROR_MEZZIO_ACQ_TIMEOUT	BFFA48D5	-1074116395
ACQIRIS_ERROR_DEVICE_ALREADY_OPEN	BFFA48D6	-1074116394
ACQIRIS_ERROR_EEPROM_CRC_FAILED	BFFA48D7	-1074116393
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_INTERRUPTED	BFFA4905	-1074116347
ACQIRIS_ERROR_WAIT_TIMEOUT	BFFA4906	-1074116346
ACQIRIS_ERROR_CLOCK_SOURCE	BFFA4907	-1074116345
ACQIRIS_ERROR_OPERATION_CANCELLED	BFFA4908	-1074116344
ACQIRIS_ERROR_FIRMWARE_NOT_AUTHORIZED	BFFA4A00	-1074116096
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_FPGA_8_LOAD	BFFA4A08	-1074116088
ACQIRIS_ERROR_FIRMWARE_NOT_SUPPORTED	BFFA4A09	-1074116087
ACQIRIS_ERROR_FPGA_1_FLASHLOAD_NO_INIT	BFFA4A10	-1074116080
ACQIRIS_ERROR_FPGA_1_FLASHLOAD_NO_DONE	BFFA4A11	-1074116079
ACQIRIS_ERROR_FPGA_2_FLASHLOAD_NO_INIT	BFFA4A12	-1074116078
ACQIRIS_ERROR_FPGA_2_FLASHLOAD_NO_DONE	BFFA4A13	-1074116077
ACQIRIS_ERROR_SELFCHECK_MEMORY	BFFA4A20	-1074116064
ACQIRIS_ERROR_SELFCHECK_DAC	BFFA4A21	-1074116063
ACQIRIS_ERROR_SELFCHECK_RAMP	BFFA4A22	-1074116062
ACQIRIS_ERROR_SELFCHECK_PCIE_LINK	BFFA4A23	-1074116061
ACQIRIS_ERROR_SELFCHECK_PCIE_DEVICE	BFFA4A24	-1074116060
ACQIRIS_ERROR_FLASH_ACCESS_TIMEOUT	BFFA4A30	-1074116048
ACQIRIS_ERROR_FLASH_FAILURE	BFFA4A31	-1074116047
ACQIRIS_ERROR_FLASH_READ	BFFA4A32	-1074116046
ACQIRIS_ERROR_FLASH_WRITE	BFFA4A33	-1074116045
ACQIRIS_ERROR_FLASH_EMPTY	BFFA4A34	-1074116044
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_ATTR_CALLBACK_STATUS	BFFA4B07	-1074115833
ACQIRIS_ERROR_ATTR_CALLBACK_EXCEPTION	BFFA4B08	-1074115832
ACQIRIS_ERROR_KERNEL_VERSION	BFFA4C00	-1074115584
ACQIRIS_ERROR_UNKNOWN_ERROR	BFFA4C01	-1074115583
ACQIRIS_ERROR_OTHER_WINDOWS_ERROR	BFFA4C02	-1074115582
ACQIRIS_ERROR_VISA_DLL_NOT_FOUND	BFFA4C03	-1074115581
ACQIRIS_ERROR_OUT_OF_MEMORY	BFFA4C04	-1074115580
ACQIRIS_ERROR_UNSUPPORTED_DEVICE	BFFA4C05	-1074115579
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 Codes	Hex value	Decimal value
ACQIRIS_ERROR_DATATYPE	BFFA4D17	-1074115305
ACQIRIS_ERROR_READMODE	BFFA4D18	-1074115304
ACQIRIS_ERROR_VM_FILE_EXTENSION	BFFA4D50	-1074115248
ACQIRIS_ERROR_VM_FILE_VERSION	BFFA4D51	-1074115247
ACQIRIS_ERROR_VM_FILE_READ	BFFA4D52	-1074115246
ACQIRIS_ERROR_VM_FILE_INVALID	BFFA4D53	-1074115245
ACQIRIS_ERROR_VM_VERIFICATION	BFFA4D54	-1074115244
ACQIRIS_ERROR_VM_CRC	BFFA4D55	-1074115243
ACQIRIS_ERROR_HW_FAILURE	BFFA4D80	-1074115200
ACQIRIS_ERROR_HW_FAILURE_CH1	BFFA4D81	-1074115199
ACQIRIS_ERROR_HW_FAILURE_CH2	BFFA4D82	-1074115198
ACQIRIS_ERROR_HW_FAILURE_CH3	BFFA4D83	-1074115197
ACQIRIS_ERROR_HW_FAILURE_CH4	BFFA4D84	-1074115196
ACQIRIS_ERROR_HW_FAILURE_CH5	BFFA4D85	-1074115195
ACQIRIS_ERROR_HW_FAILURE_CH6	BFFA4D86	-1074115194
ACQIRIS_ERROR_HW_FAILURE_CH7	BFFA4D87	-1074115193
ACQIRIS_ERROR_HW_FAILURE_CH8	BFFA4D88	-1074115192
ACQIRIS_ERROR_HW_FAILURE_EXT1	BFFA4DA0	-1074115168
ACQIRIS_ERROR_MAC_T0_ADJUSTMENT	BFFA4DC0	-1074115136
ACQIRIS_ERROR_MAC_ADC_ADJUSTMENT	BFFA4DC1	-1074115135
ACQIRIS_ERROR_MAC_RESYNC_ADJUSTMENT	BFFA4DC2	-1074115134
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
ACQIRIS_WARN_READPARA_FLAGS_ADAPTED	3FFA4E15	1073368597
ACQIRIS_WARN_SIMOPTION_STRING_UNKNOWN	3FFA4E16	1073368598
ACQIRIS_WARN_INSTRUMENT_IN_USE	3FFA4E17	1073368597
ACQIRIS_WARN_HARDWARE_TIMEOUT	3FFA4E60	1073368672
ACQIRIS_WARN_RESET_IGNORED	3FFA4E61	1073368671
ACQIRIS_WARN_SELFCHECK_MEMORY	3FFA4F00	1073368832
ACQIRIS_WARN_CLOCK_SOURCE	3FFA4F01	1073368833
ACQIRIS_WARN_NUMERIC_OVERFLOW	3FFA4F20	1073368864

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

The API has been split into three families:

- Acqrs Generic functions - AqBx - these can be used for all Acqiris Instruments
- AcqrsD1 Digitizer functions - AqDx - to be used for Digitizers and Analyzers
- AcqrsT3 Time-to-Digital Converter functions - AqTx - to be used for the family of Time-to-Digital Converters

All of these functions are still contained in one library called **AqDrv4**. However, there are separate files for the headers and the LabWindows front-panel interface. The LabView interface is also split into the three corresponding AqXX parts. The AcqrsD1 section includes redundant copies of the generic functions so that backward calling compatibility can be maintained for existing code.

Visual Basic support will be limited to the Generic and AcqrsD1 families. Time-to-Digital Converters are not supported.

AcqirisInterface.h is the header file for these functions:

Generic Initialization Functions	*Function Name*
Number of Physical Instruments	`Acqrs_getNbrInstruments`
Initialization	`Acqrs_init`
Initialization with Options	`Acqrs_InitWithOptions`
Simulation Options	`Acqrs_setSimulationOptions`
Generic Calibration Functions	
Calibrate Instrument	`Acqrs_calibrate`
Calibrate Instrument Extended	`Acqrs_calibrateEx`
Interrupt Calibration	`Acqrs_calibrateCancel`
Load calibration values from a file	`Acqrs_calLoad`
Query about the necessity of self calibration	`Acqrs_calRequired`
Save all calibration values in a file	`Acqrs_calSave`
Generic Query Functions	
Instrument Basic Data	`Acqrs_getInstrumentData`
Instrument Information	`Acqrs_getInstrumentInfo`
Number of Channels	`Acqrs_getNbrChannels`
Generic Utility Functions	
Version	`Acqrs_getVersion`
Error Message	`Acqrs_errorMessage`
Reset	`Acqrs_reset`
Set LED Color	`Acqrs_setLEDColor`
Close an instrument	`Acqrs_close`
Close all instruments	`Acqrs_closeAll`
Resume the control of an instrument that was suspended	`Acqrs_resumeControl`
Suspend control of an instrument	`Acqrs_suspendControl`
Prepare for entry or return from the system power down state	`Acqrs_powerSystem`

AcqirisD1Interface.h is the header file for these functions:

Digitizer Initialization Functions	*Function Name*
Number of Physical Instruments (deprec.)	`AcqrsD1_getNbrPhysicalInstruments`
MultiInstrument Auto Define	`AcqrsD1_multiInstrAutoDefine`

Initialization (deprec.)	AcqrsD1_init
Initialization with Options (deprec.)	AcqrsD1_InitWithOptions
Simulation Options (deprec.)	AcqrsD1_setSimulationOptions
Digitizer Calibration Functions	
Calibrate Instrument (deprec.)	AcqrsD1_calibrate
Calibrate Instrument Extended (deprec.)	AcqrsD1_calibrateEx
Digitizer 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 Memory Settings (extended)	AcqrsD1_configMemoryEx
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 AcqrsD1_configAvgConfigInt32 AcqrsD1_configAvgConfigReal64
Configure (program) on-board FPGA (deprec.)	AcqrsD1_configLogicDevice
Configure Array of Setup Parameters	AcqrsD1_configSetupArray
Logical Device IO	AcqrsD1_logicDeviceIO
MultiInstrument Manual Define	AcqrsD1_multiInstrDefine
MultiInstrument Undefine	AcqrsD1_multiInstrUndefineAll
Setup Streaming in SC Analyzer	AcqrsD1_setAttributeString
Digitizer 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
Digitizer Data Transfer Functions	
Universal Waveform Read	AcqrsD1_readData
Accumulate Data	AcqrsD1_accumulateData
Averaged Data	AcqrsD1_averagedData

Read Frequency Counter	`AcqrsD1_readFCounter`
Digitizer Query Functions	
Query External Clock	`AcqrsD1_getExtClock`
Query Horizontal Settings	`AcqrsD1_getHorizontal`
Query Channel Combination	`AcqrsD1_getChannelCombination`
Query Memory Settings	`AcqrsD1_getMemory`
Query Memory Settings (extended)	`AcqrsD1_getMemoryEx`
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`
	`AcqrsD1_getAvgConfigInt32`
	`AcqrsD1_getAvgConfigReal64`
Instrument Basic Data (deprec.)	`AcqrsD1_getInstrumentData`
Instrument Information (deprec.)	`AcqrsD1_getInstrumentInfo`
Number of Channels	`AcqrsD1_getNbrChannels`
Query Array of Setup Parameters	`AcqrsD1_getSetupArray`
Digitizer 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`
Digitizer Utility Functions	
Best Nominal Samples	`AcqrsD1_bestNominalSamples`
Best Sampling Interval	`AcqrsD1_bestSampInterval`
Version	`AcqrsD1_getVersion`
Error Message	`AcqrsD1_errorMessage`
Extended Error Message	`AcqrsD1_errorMessageEx`
Reset (deprec.)	`AcqrsD1_reset`
Reset Digitizer Memory	`AcqrsD1_resetDigitizerMemory`
Restore Internal Registers	`AcqrsD1_restoreInternalRegisters`
Set LED Color	`AcqrsD1_setLEDColor`
Close all instruments (deprec.)	`AcqrsD1_closeAll`

AcqirisT3Interface.h is the header file for these functions:

Time-to-Digital Converter Configuration Functions

Configure Acquisition Conditions	`AcqrsT3_configAcqConditions`
Configure Channel	`AcqrsT3_configChannel`

Time-to-Digital Converter Acquisition Control Functions

Start Acquisition	`AcqrsT3_acquire`
Query Acquisition Status	`AcqrsT3_acqDone`
Force trigger	`AcqrsT3_forceTrig`
Stop Acquisition	`AcqrsT3_stopAcquisition`
Wait for End of Acquisition	`AcqrsT3_waitForEndOfAcquisition`

Time-to-Digital Converter Data Transfer Functions

Universal Time Data Read	`AcqrsT3_readData`
	`AcqrsT3_readDataInt32`
	`AcqrsT3_readDataReal64`

Time-to-Digital Converter Query Functions

Query Acquisition Conditions	`AcqrsT3_getAcqConditions`
Query Channel	`AcqrsT3_getChannel`

2.3. API Function descriptions

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

2.3.1 Acqrs_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 = Acqrs_calibrate(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Calibrate Instrument.vi
```

Visual Basic .NET Representation

```
Acqrs_calibrate (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_calibrate(instrumentID)
```

2.3.2 Acqrs_calibrateCancel

Purpose

Interrupts a calibration of the instrument launched from a different thread

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 = Acqrs_calibrateCancel(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Calibrate Cancel.vi
```

Visual Basic .NET Representation

```
Acqrs_calibrateCancel (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_calibrateCancel(instrumentID)
```

2.3.3 Acqrs_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 **Acqrs_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 3.16.2, **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, DC271A, DC271AR and 10-bit-FAMILY 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. It will use the new values that have been asked for.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) CalibrateEx Instrument.vi
```

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status]= Aq_calibrateEx(instrumentID, calType, modifier, flags)
```

2.3.4 Acqrs_calLoad

Purpose

Load calibration values from file. (For 10-bit-FAMILY//U1071A-FAMILY)

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
filePathName	ViConstString	File path and file name
flags	ViInt32	Flags, may be: 0 = default filename. Calibration values will be loaded from the 'snXXXXX_calVal.bin' file in the working directory. 'filePathName' MUST be NULL or "" (empty String). 1 = specify path only. Calibration values will be loaded from the 'snXXXXX_calVal.bin' file in the specified directory. 'filePathName' MUST be non-NULL. 2 = specify filename. 'filePathName' represents the filename (with or without path) and MUST be non-NULL and non-empty.

Return Value

Name	Type	Description
status	ViStatus	Refer to chapter 2.1 in Programmer's Reference Manual for error codes.

Discussion

Load calibration values from a binary file. The path or full filename can be specified, else default values will be used ('snXXXXX_calVal.bin' file in the working directory).

The function can return the following error codes:

- ACQIRIS_ERROR_FILE_CORRUPTED if the file is corrupted
- ACQIRIS_ERROR_FILE_VERSION if the file has been generated with a driver version different than the used one (major and minor).
- ACQIRIS_ERROR_FILE_SERIAL if the file does not correspond to the instrument or an AS bus multi-instrument has changed.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_calLoad(ViSession instrumentID, ViConstString
                    filePathName, ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Calibration Load Instrument.vi
```

Visual Basic .NET Representation

```
Acqrs_calLoad (ByVal instrumentID As Int32, _
             ByVal filePathName As String, _
             ByVal flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_calLoad(instrumentID, filePathName, flags)
```

2.3.5 Acqrs_calRequired

Purpose

Check if a self calibration is needed. (For 10-bit-FAMILY/U1071A-FAMILY)

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	Channel number [0,1… Nchan]

Output

Name	Type	Description
isRequiredP	ViBoolean	= VI_TRUE if a calibration on channel *chan* is needed VI_FALSE otherwise

Return Value

Name	Type	Description
status	ViStatus	Refer to chapter 2.1 in Programmer's Reference Manual for error codes.

Discussion

Query about the necessity of self calibration.

The value *channel = 0* can be used to do the query on all channels simultaneously.

A calibration is needed for channel *channel, > 0,* if one or more of the 3 following condition is true:

- The channel *channel* of the instrument has never been calibrated for the desired acquisition conditions.
- It has been calibrated more than 2 hours ago.
- The instrument temperature since the last calibration has changed by more than 5°C.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_calRequired(ViSession instrumentID, ViInt32 channel,
                    ViBoolean* isRequiredP);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Query Calibration Required.vi
```

Visual Basic .NET Representation

```
Acqrs_calRequired (ByVal instrumentID As Int32, ByVal channel As Int32,_
   ByRef isRequired As Boolean) As Int32
```

MATLAB MEX Representation

```
[status isRequired] = Aq_calRequired(instrumentID, channel)
```

2.3.6 Acqrs_calSave

Purpose

Save all calibration values in a binary file. (For 10-bit-FAMILY//U1071A-FAMILY)

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
filePathName	ViConstString	File path and file name
flags	ViInt32	Flags, may be: 0 = default filename. Calibration values will be loaded from the 'snXXXXX_calVal.bin' file in the working directory. 'filePathName' MUST be NULL or "" (empty String). 1 = specify path only. Calibration values will be loaded from the 'snXXXXX_calVal.bin' file in the specified directory. 'filePathName' MUST be non-NULL. 2 = specify filename. 'filePathName' represents the filename (with or without path) and MUST be non-NULL and non-empty.

Return Value

Name	Type	Description
status	ViStatus	Refer to chapter 2.1 in Programmer's Reference Manual for error codes.

Discussion

Write calibration values in a binary file. The path or full filename can be specified, else default values will be used ('snXXXXX_calVal.bin' file in the working directory).

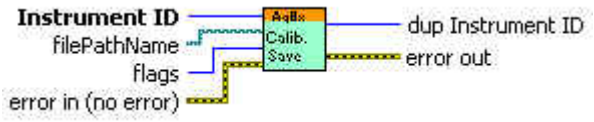
NOTE: If the file already exists, it will be overwritten.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_calSave(ViSession instrumentID, ViConstString
                    filePathName, ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Calibration Save.vi
```



Visual Basic .NET Representation

```
Acqrs_calSave (ByVal instrumentID As Int32, _
             ByVal filePathName As String, _
             ByVal flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_calSave(instrumentID, filePathName, flags)
```

2.3.7 Acqrs_close

Purpose

Closes 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

Close the specified instrument. Once closed, this instrument is not available anymore and needs to be reenabled using 'InitWithOptions' or 'init'. 10-bit-FAMILY digitizers will have their power consumption lowered. Appropriate warm-up time may be needed when they are used again.

For freeing properly all resources, 'closeAll' must still be called when the application closes, even if 'close' was called for each instrument.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Close.vi
```

Visual Basic .NET Representation

```
Acqrs_close (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_close(instrumentID)
```

2.3.8 Acqrs_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. 10-bit-FAMILY digitizers will have their power consumption lowered. Appropriate warm-up time may be needed when they are used again.

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 = Acqrs_closeAll(void);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Close All Instruments.vi
```

Visual Basic .NET Representation

```
Acqrs_closeAll ( ) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_closeAll()
```

2.3.9 Acqrs_configLogicDevice

Purpose

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

NOTE: With the exception of AC and SC Analyzers, this function now needs to be used only by VxWorks users to specify the filePath for FPGA .bit files. Otherwise it should no longer have to be used

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
deviceName	ViChar [ ]	Identifies which device to program For the AC210/AC240 and SC210/SC240 modules this string must be "Block1Dev1". Alternatively it can be "ASBUS::n::Block1Dev1" with n ranging from 0 to the number of modules -1. When clearing the FPGA's, the string must be "Block1DevAll".
filePathName	ViChar [ ]	File path and file name
flags	ViInt32	flags, 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 3 = 0 + use normal search order with AqDrv4.ini file

Return Value

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

Discussion

With flags = 2 in VxWorks systems, the filePathName must point to a directory containing the FPGA configuration files with extension '.bit'

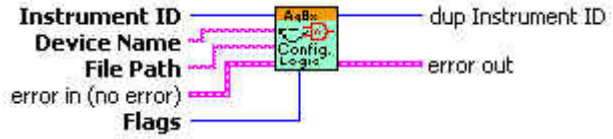
With flags = 0 or 3, the filePathName must point to an FPGA configuration file with extension '.bit', e.g. "D:\Averagers\FPGA\AP100DefaultFPGA1.bit".
For more details on programming on-board logic devices, please refer to the **Programmer's Guide** sections 3.2, **Device Initialization** and 3.3, **Device Configuration**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Configure Logic Device.vi
```



Visual Basic .NET Representation

```
Acqrs_configLogicDevice (ByVal instrumentID As Int32, _
                  ByVal deviceName As String, _
                  ByVal filePathName As String, _
                  ByVal flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_configLogicDevice(instrumentID, deviceName, filePathName, flags)
```

2.3.10 Acqrs_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
errorMessageSize	ViInt32	Size of the errorMessage string in bytes (suggested size 512)

Output

Name	Type	Description
errorMessage	ViChar [ ]	Pointer to user-allocated string (suggested size 512) into which the error-message text is returned

Return Value

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

Discussion

This function should be called immediately after the return of the error status to ensure that the additional information remains available. For file errors, the returned message will contain the file name and the original 'ansi' error string. This is particularly useful for calls to the following functions:

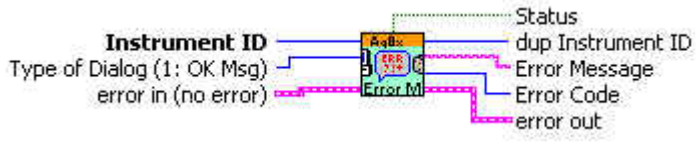
Acqrs_calibrate	Acqrs_calibrateEx
Acqrs_configLogicDevice	Acqrs_configMode
Acqrs_init	Acqrs_InitWithOptions

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_errorMessage(ViSession instrumentID,
                     ViStatus errorCode, ViChar errorMessage[],
                     ViInt32 errorMessageSize);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Error Message.vi
```



Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status errorMessage]= Aq_errorMessage(instrumentID, errorCode)
```

2.3.11 Acqrs_getDevType

Purpose

Returns the deviceType which indicates which family of the API functions can be used.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
devTypeP	ViInt32*	Pointer to a device type (see AqDevType) with 1 = Digitizer (AcqrsD1) 2 = RC2xx Generator (AcqrsG2) 4 = TC Time-to-Digital Converter (AcqrsT3)

Return Value

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

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_getDevType(ViSession instrumentID,
                    ViInt32* devTypeP);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx)Query Device Type.vi
```

Visual Basic .NET Representation

```
Acqrs_getDevType (ByVal instrumentID As Int32, _
             ByRef devType As Long) As Int32
```

MATLAB MEX Representation

```
[status devType]= Aq_getDevType(instrumentID)
```

2.3.12 Acqrs_getDevTypeByIndex

Purpose

Returns the deviceType which indicates which family of API functions can be used.

Parameters

Input

Name	Type	Description
devIndex	ViInt32	Device Index (the integer part of the resource name as used in **Acqrs_initWithOptions**. See the **Programmer's Guide** section 3.2.1)

Output

Name	Type	Description
devTypeP	ViInt32*	Pointer to a device type (see AqDevType) with 1 = Digitizer (AcqrsD1) 2 = RC2xx Generator (AcqrsG2) 4 = TC Time-to-Digital Converter (AcqrsT3)

Return Value

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

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_getDevTypeByIndex(ViInt32 devIndex,
                    ViInt32* devTypeP);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx)Query Device Type By Index.vi
```

Visual Basic .NET Representation

```
Acqrs_ getDevTypeByIndex (ByVal devIndex As Int32, _
                  ByRef devType As Long) As Int32
```

MATLAB MEX Representation

```
[status devType]= Aq_getDevType(devIndex)
```

2.3.13 Acqrs_getInstrumentData

Purpose

Returns some basic data about a specified instrument.

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 module.
busNbr	ViInt32	Bus number of the module location.
slotNbr	ViInt32	Slot number of the module location. (logical)

Return Value

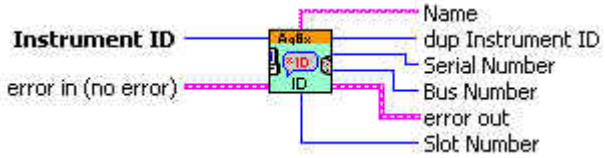
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Query Instrument ID.vi
```



Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status name serialNbr busNbr slotNbr]= Aq_getInstrumentData(instrumentID)
```

2.3.14 Acqrs_getInstrumentInfo

Purpose

Returns general information about a specified instrument.

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	Returns 1 if the *m*'th module of a multi-instrument is the master, 0 otherwise. *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.
`"LogDevDataLinks"`	ViInt32	Number of available data links for a streaming analyzer
`"LOGDEVHDRBLOCK`*m*`DEV`*n*`S`*string*`"`	ViChar[ ]	Returns information about FPGA firmware loaded. See comments below.
`"MaxSamplesPerChannel"`	ViInt32	Maximum 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 (not connected through AS bus) 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.

Parameter String	Returned Type	Description
		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
"SSRTimeStamp"	ViReal64	Current value of time stamp for Analyzers in SSR mode.
"TbNextSegmentPad"	ViInt32	Returns the additional array space (in samples) per segment needed for the image read of **AcqrsD1_readData**. It concerns the data available after the next call to AcqrsD1_acquire, as opposed to any current or past acquisition with different conditions.
"TbSegmentPad"	ViInt32	Returns the additional array space (in samples) per segment needed for the image read of **AcqrsD1_readData**. It concerns the current data available, as opposed to any future acquisition with different conditions.
"Temperature *m*"	ViInt32	Temperature in degrees Centigrade (^{o}C)
"TrigLevelRange *chan*"	ViReal64	Trigger Level Range on channel *chan*
"VersionUserDriver"	ViChar[ ]	String containing the full driver version.

Discussion

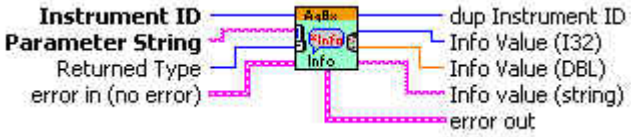
The case of "LOGDEVHDRBLOCK*m*DEV*n*S *string*" is one in which several possible values of *m, n,* and *string* are allowed. The single digit number *m* refers to the FPGA block in the module. For the moment this must always have the value 1. The single digit number *n* refers to the FPGA device in the 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".

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) 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 .NET Representation

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

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

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

MATLAB MEX Representation

```
[status infoValue] = Aq_getInstrumentInfo(instrumentID, parameterString,
                    dataTypeString)
```

Allowed values of dataTypeString are `'integer'`, `'double'`, or `'string'`.

2.3.15 Acqrs_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 = Acqrs_getNbrChannels(ViSession instrumentID, ViInt32*
                    nbrChannels);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Query Number of Channels.vi
```

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status nbrChannels] = Aq_getNbrChannels(instrumentID)
```

2.3.16 Acqrs_getNbrInstruments

Purpose

Returns the number of Acqiris instruments found on the computer.

Parameters

Output

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

Return Value

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

Discussion

In the case of multiple processes accessing the Agilent Acqiris instruments, this function will return the number of currently available instruments. If an instrument has already been initialized in another process, it will not be available unless it has been suspended via a call to Acqrs_suspendControl.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Query Number of Instruments.vi
```

Visual Basic .NET Representation

```
Acqrs_getNbrInstruments (ByRef nbrInstruments As Int32) As Int32
```

MATLAB MEX Representation

```
[status nbrInstruments]= Aq_getNbrInstruments()
```

2.3.17 Acqrs_getVersion

Purpose

Returns version numbers associated with a specified instrument 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 Instrument 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

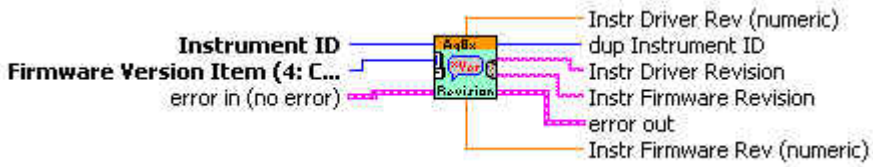
For drivers, the version number is composed of 2 parts. The upper 2 bytes represent the major version number, and the lower 2 bytes represent the minor version number.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Revision Query.vi
```



Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status version] = Aq_getVersion(instrumentID, versionItem)
```

2.3.18 Acqrs_init

Purpose

Initializes an instrument.

Parameters

Input

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

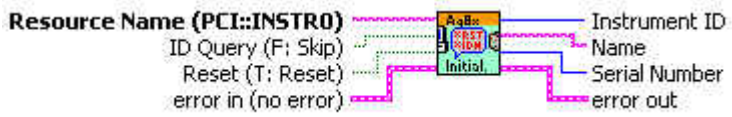
The function returns the error code ACQIRIS_ERROR_INIT_STRING_INVALID when the initialization string could not be interpreted.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Initialize.vi
```



Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status instrumentID] = Aq_init(instrumentID, IDQuery, resetDevice)
```

2.3.19 Acqrs_InitWithOptions

Purpose

Initializes an instrument with options.

Parameters

Input

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

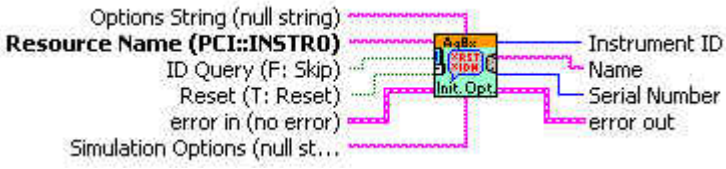
Multiple options can be given; Separate the option=value pairs with ‘,’ characters.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Initialize with Options.vi
```



Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status instrumentID]= Aq_initWithOptions(resourceName, IDQuery, resetDevice,
                      optionsString)
```

2.3.20 Acqrs_logicDeviceIO

Purpose

Reads/writes a number of 32-bit data values from/to a user-defined register in on-board logic devices, such as user-programmable FPGAs. It is useful for AC/SC Analyzers only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
deviceName	ViChar [ ]	Identifies which device to read from or write to. For the AC210/AC240 and SC210/SC240 modules this string must be "Block1Dev1". Alternatively it can be "ASBUS::n::Block1Dev1" with n ranging from 0 to the number of modules -1
registerID	ViInt32	Register Number, can typically assume 0 to 127
nbrValues	ViInt32	Number of data values to read
dataArray	ViInt32 [ ]	User-supplied array of data values
readWrite	ViInt32	Direction 0 = read from device, 1 = write to device
flags	ViInt32	Currently unused, set to “0”

Return Value

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

Discussion

This function is only useful if the user programmed the on-board logic device (FPGA).

Typically, *nbrValues* is set to 1, but it may be larger if the logic device supports internal address auto-incrementation. The following example reads the (32-bit) contents of register 5 to *reg5Value*:

```
ViStatus status =
     Acqrs_logicDeviceIO(ID, "Block1Dev1", 5, 1, &reg5Value, 0, 0);
```

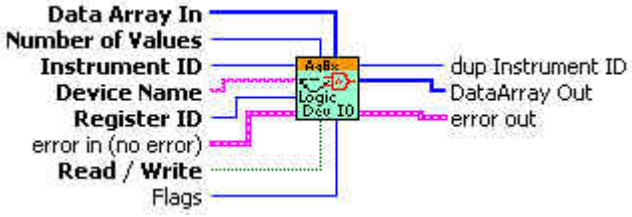
Note that *dataArray* must always be supplied as an address, even when writing a single value.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_logicDeviceIO(ViSession instrumentID,
                      ViChar deviceName[], ViInt32 registerID,
                      ViInt32 nbrValues,   ViInt32 dataArray[],        ViInt32
                      readWrite,   ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Logic Device IO.vi
```



Visual Basic .NET Representation

```
Acqrs_logicDeviceIO (ByVal instrumentID As Int32, _
                        ByVal deviceName As String, _
                        ByVal registerID As Int32, _
                        ByVal nbrValues As Int32, _
                        ByRef dataArray As Int32, _
                        ByVal readWrite As Int32, _
                        ByVal modifier As Int32) As Int32
```

MATLAB MEX Representation

Because of the separation of input and output arguments in MATLAB two functions are needed:

```
[status dataArray] = Aq_logicDeviceRead(instrumentID, deviceName, registerID,
                      nbrValues, modifier)
[status] = Aq_logicDeviceWrite(instrumentID, deviceName, registerID,
                      nbrValues, dataArray, modifier)
```

2.3.21 Acqrs_powerSystem

Purpose

Forces all instruments to prepare entry into or return from the system power down state.

Parameters

Input

Name	Type	Description
state	ViInt32	0 = 'AqPowerOff' of the AqPowerState enum 1 = 'AqPowerOn' of the AqPowerState enum
flags	ViInt32	Currently unused, set to "0"

Return Value

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

Discussion

Typically, this function is called by a 'Power Aware' application, when it catches a 'system power down' event, such as 'hibernate'.

If 'state == 0', it will suspend all other calling threads. If a thread is performing a long operation which cannot be completed within milliseconds, such as 'calibrate', it will be interrupted immediately and will get the status 'ACQIRIS_ERROR_OPERATION_INTERRUPTED'. Note that if an acquisition is still running while Acqrs_powerSystem(0, 0) is called, it might be incomplete or corrupted.

If 'state == 1', it will reenable the instruments at the same state as they were before Acqrs_powerSystem(0, 0). Threads which were suspended will be resumed. However, interrupted operations which returned an error 'ACQIRIS_ERROR_OPERATION_INTERRUPTED' have to be redone.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_powerSystem(ViInt32 state, ViInt32 flags);
```

LabVIEW Representation

There is no LabVIEW implementation of this function.

Visual Basic .NET Representation

```
Acqrs_powerSystem(ByVal state As Int32, ByVal flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_powerSystem(state, flags)
```

2.3.22 Acqrs_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 = Acqrs_reset(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Reset.vi
```

Visual Basic .NET Representation

```
Acqrs_reset (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_reset(instrumentID)
```

2.3.23 Acqrs_resetMemory

Purpose

Resets the instrument's 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 = Acqrs_resetMemory(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Reset Memory.vi
```

Visual Basic .NET Representation

```
Acqrs_resetMemory (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_resetMemory(instrumentID)
```

2.3.24 Acqrs_resumeControl

Purpose

Resume the control of an instrument that was suspended (see **Acqrs_suspendControl**).

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to chapter 2.1 in Programmer's Reference Manual for error codes.

Discussion

This function reacquires the driver lock of the instrument and allows calls to it from the current process. The error code ACQIRIS_ERROR_DEVICE_ALREADY_OPEN is returned when calling an instrument already locked by another process.

After successfully calling *Acqrs_resumeControl*, the module will be set to a default hardware state. It will have no valid data and the timestamp will be set to 0. When the next acquisition is started, the module will be configured with all of the unmodified settings from before the *Acqrs_suspendControl* was invoked.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Resume Control.vi
```

Visual Basic Representation

```
ResumeControl (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
Acqrs_ resumeControl (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_resumeControl(instrumentID)
```

2.3.25 Acqrs_setAttributeString

Purpose

Sets an attribute with a string value (for use in SC Streaming Analyzers ONLY).

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
name	ViConstString	ASCII string that specifies options "odlTxBitRate" is currently the only one used
value	ViConstString	For "odlTxBitRate" can have values like "2.5G","2.125G", or "1.0625G"

Return Value

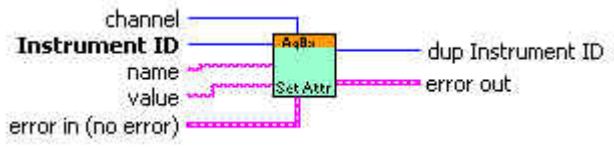
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = Acqrs_setAttributeString(ViSession instrumentID,
                      ViInt32 channel, ViConstString name,
                      ViConstString value);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Set Attribute String.vi
```



Visual Basic .NET Representation

```
Acqrs_setAttributeString (ByVal instrumentID As Int32, _
                      ByVal channel As Int32, _
                      ByVal name As String, _
                      ByVal value As String) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_setAttributeString (instrumentID, channel, name, value)
```

2.3.26 Acqrs_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 = Acqrs_setLEDColor(ViSession instrumentID,
                    ViInt32 color);
```

LabVIEW Representation

```
Acqiris Bx.lvlib: (or Aq Bx) Set LED Color.vi
```

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status ] = Aq_setLEDColor(instrumentID, color)
```

2.3.27 Acqrs_setSimulationOptions

Purpose

Sets one or several options which will be used by the function **Acqrs_InitWithOptions**, provided that the **optionsString** supplied with that function 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 3.2.10, **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 = Acqrs_setSimulationOptions(
                    ViString simOptionString);
```

LabVIEW Representation

```
Use Acqiris Bx.lvlib: (or Aq Bx) Initialize with Options.vi
```

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status] = Aq_setSimulationOptions(simOptionsString)
```

2.3.28 Acqrs_suspendControl

Purpose

Suspend control of an instrument to allow using it from another process. NOTE: This is only available for Windows operating systems.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Return Value

Name	Type	Description
status	ViStatus	Refer to chapter 2.1 in Programmer's Reference Manual for error codes.

Discussion

This function releases the driver lock of the instrument and prevents all further calls from the current process. The error code ACQIRIS_ERROR_INVALID_HANDLE is returned when calling functions on a suspended instrument. Use **Acqrs_resumeControl** to reacquire the control of the instrument.

Once suspended, this instrument can be used from another process. However, if this is the first time this other process is used, all desired acquisition settings must be defined and a calibration will be needed.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Suspend Control.vi
```

Visual Basic Representation

```
SuspendControl (ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
Acqrs_suspendControl (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_suspendControl(instrumentID)
```

2.3.29 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
```

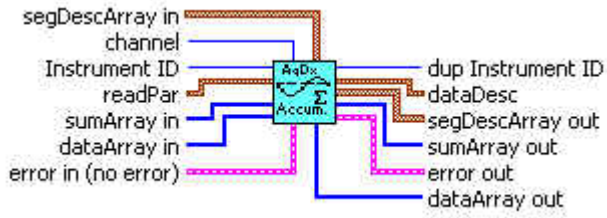
LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status dataDesc segDescArray dataArray sumArray]=
         AqD1_accumulateData(instrumentID, channel, readPar)

Note: The older form Aq_accumulateData is deprecated.
     Please convert to the newer version.
```

2.3.30 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status done]= AqD1_acqDone(instrumentID)

Note: The older form Aq_acqDone is deprecated.
     Please convert to the newer version.
```

2.3.31 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_acquire(instrumentID)

Note: The older form Aq_acquire is deprecated.
     Please convert to the newer version.
```

2.3.32 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	= 0, normal = 4, resets the time stamp counter (U1071A & 10-bit-Family only)
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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_acquireEx(instrumentID, acquireMode, acquireFlags,
                      acquireParams, reserved)
Note: The older form Aq_acquireEx is deprecated.
     Please convert to the newer version.
```

2.3.33 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.

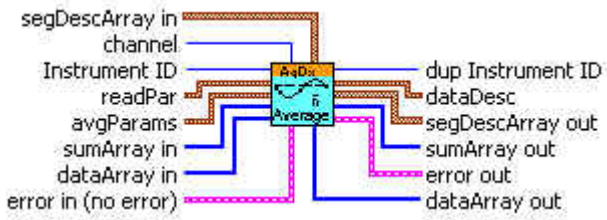
LabWindowsCVI/Visual C++ Representation

```
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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status dataDesc segDescArray dataArray sumArray]=
              AqD1_averagedData(instrumentID, channel, readPar, nbrAcq,
                     calculateMean, timeout)
Note: The older form Aq_averagedData is deprecated.
     Please convert to the newer version.
```

2.3.34 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status nomSamples]= AqD1_bestNominalSamples(instrumentID)

Note: The older form Aq_bestNominalSamples is deprecated.
     Please convert to the newer version.
```

2.3.35 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 requested 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.

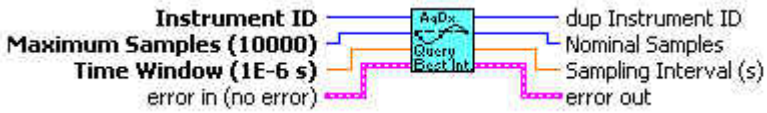
NOTE: The returned "recommended" values for the sampling interval **sampInterval** and the nominal number of samples **nomSamples** are expected to be used for configuring the instrument with calls to **AcqrsD1_configMemory** and **AcqrsD1_configHorizontal**. Make sure to use the same number of segments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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status sampInterval nomSamples]= AqD1_bestSampInterval(instrumentID,
                      maxSamples, timeWindow)

Note: The older form Aq_bestSampInterval is deprecated.
     Please convert to the newer version.
```

2.3.36 AcqrsD1_calibrate (DEPRECATED)

Purpose

Performs an auto-calibration of the instrument. See **Acqrs_calibrate**.

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

Please refer to **Acqrs_calibrate**

Visual Basic Representation

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

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status]= Aq_calibrate(instrumentID)
```

2.3.37 AcqrsD1_calibrateEx (DEPRECATED)

Purpose

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

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 3.16.2, **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, DC271A, DC271AR and 10-bit-FAMILY 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. It will use the new values that have been asked for.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

See **Acqrs_calibrateEx**

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
```

MATLAB MEX Representation

```
[status]= Aq_calibrateEx(instrumentID, calType, modifier, flags)
```

2.3.38 AcqrsD1_close (DEPRECATED)

Purpose

Closes an instrument. See **Acqrs_close**

.

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

Close the specified instrument. Once closed, this instrument is not available anymore and needs to be reenabled using 'InitWithOptions' or 'init'.
For freeing properly all resources, 'closeAll' must still be called when the application closes, even if 'close' was called for each instrument.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_close(void);
```

LabVIEW Representation

See **Acqrs_close**

Visual Basic Representation

```
Close(ByVal instrumentID As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_close (ByVal instrumentID As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_close(instrumentID)
```

2.3.39 AcqrsD1_closeAll (DEPRECATED)

Purpose

Closes all instruments in preparation for closing the application. See **Acqrs_closeAll**.

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

See **`Acqrs_closeAll`**.

Visual Basic Representation

```
CloseAll ( ) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_closeAll ( ) As Int32
```

MATLAB MEX Representation

```
[status]= Aq_closeAll()
```

2.3.40 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.
"FixedSamples"	ViInt32	For Threshold Gate type in **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. Number of samples transmitted for each point over threshold. It must be a multiple of 4. 0 = No limit imposed.
"GateType"	ViInt32	For **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. 1 = User Gates 2 = Threshold Gates
"HistoTDCEnable"	ViInt32	For **AP240/AP235 Averagers** ONLY. 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).
"NbrMaxGates"	ViInt32	For Threshold Gate type in **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. Maximum number of gates allowed for each segment. 0 = No limit imposed
"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** and **Peak^{TDC}** ONLY.
"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.

Parameter String	Data Type	Description
"NoiseBase"	ViReal64	Value in Volts of the value to be added in Noise Supressed Averaging. For **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
"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 SSR** and **Peak^{TDC} 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.
"PreSamples"	ViInt32	For **AP240/AP235 SSR** and **Peak^{TDC} 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.
"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).
"StartDeltaNegPeak"	ViInt32	Negative excursion needed before searching for negative peak. For **AP101/AP201 Analyzers** ONLY.
"StartDeltaPosPeak"	ViInt32	Positive excursion needed before searching for positive peak. May assume values between 1 and 0xff. For **AP101/AP201 Analyzers** ONLY.
"StartDeltaPosPeakV"	ViReal64	Positive excursion needed before searching for positive peak. Must be positive. For **Peak^{TDC} mode Analyzers** ONLY.
"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.**
"StopDelay "	ViInt32	Stop delay in samples. May assume values between 0 and 1048560 (20971201048560) in steps of of 16 (32) as explained below. The limit is StepSize*(64*1024-1)
"TdcHistogramDepth"	ViInt32	The depth of the histogram for **Peak^{TDC}** mode. 0 means 16-bit accumulation bins. 1 means 32-bit accumulation bins.
"TdcHistogramHorzRes"	ViInt32	The horizontal resolution of the histogram for interpolated peaks in the **Peak^{TDC}** mode. 0 means that each bin corresponds to a sampling interval. ≤4 means that each bin corresponds to ½**n of a sampling interval.

Parameter String	Data Type	Description
"TdcHistogramIncrement"	ViInt32	The desired increment to be applied for each entry; 1 means increment by 1, for **SimpleTDC Averager** and **Peak^{TDC} Analyzer** modes ONLY. 2 means increment by the ADCvalue – NoiseBase for a **SimpleTDC Averager** and by the ADCvalue for the **Peak^{TDC} Analyzer**
"TdcHistogramMode"	ViInt32	The type of histogram for **Peak^{TDC}** mode ONLY. 0 means no histogram. Data only is available for each acquisition. 1 for a histogram.
"TdcHistogramVertRes"	ViInt32	The vertical resolution of the histogram for interpolated peaks when the **TDCHistogramIncrement** is 2 in the **Peak^{TDC}** mode. 0 means that each bin corresponds to a sampling interval. ≤4 means that each bin corresponds to ½**n of a sampling interval.
"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.
"TdcOverlaySegments"	ViInt32	This option controls the horizontal binning of data in the **Peak^{TDC}** histogram mode. 0 means that each segment will be histogrammed independently. 1 means that all segments will be histogrammed on a common time axis.
"TdcProcessType"	ViInt32	The desired processing for **Peak^{TDC}** mode peak finding. May assume 0 = No processing 1 = Standard peak finding (no interpolation) 2 = Interpolated peaks 3 = 8 sample peak regions for data readout 4 = 16 sample peak regions for data readout
"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** or **Peak^{TDC}** with **Threshold Gates**.
"TrigAlways"	ViInt32	May assume 0 (no trigger output) and 1 (trigger output on), in the case of no acquisition.
"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.
"TrigResync"	ViInt32	May assume 0 (no resync), 1 (resync) and 2 (free run)
"ValidDeltaNegPeak"	ViInt32	Positive excursion needed to validate a negative peak. May assume values between 1 and 0xff. For **AP101/AP201** ONLY.
"ValidDeltaPosPeak"	ViInt32	Negative excursion needed to validate a positive peak. May assume values between 1 and 0xff. For **AP101/AP201** ONLY.
"ValidDeltaPosPeakV"	ViReal64	Negative excursion needed to validate a positive peak. Must be positive. For **Peak^{TDC}** mode **Analyzers** 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.

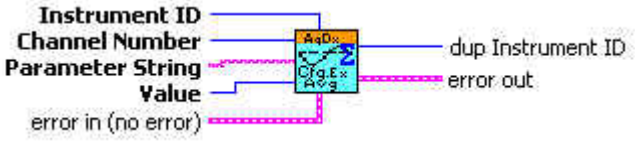
Note that this function takes the **address**, not the value of the parameter to be set.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Extended Configure Averager.vi
      This Vi is polymorphic, the value can be either I32 or DBL.
```



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
```

MATLAB MEX Representation

```
Note: Please see AqD1_configAvgConfigInt32 and AqD1_configAvgConfigReal64.
```

2.3.41 AcqrsD1_configAvgConfigInt32

Purpose

Configures a long 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	ViInt32	Value to set.

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.
"FixedSamples"	ViInt32	For Threshold Gate type in **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. Number of samples transmitted for each point over threshold. It must be a multiple of 4. 0 = No limit imposed.
"GateType"	ViInt32	For **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. 1 = User Gates 2 = Threshold Gates
"HistoTDCEnable"	ViInt32	For **AP240/AP235 Averagers** ONLY. 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).
"NbrMaxGates"	ViInt32	For Threshold Gate type in **AP240/AP235 Analyzers** and **Peak^{TDC}** ONLY. Maximum number of gates allowed for each segment. 0 = No limit imposed
"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** and **Peak^{TDC}** ONLY.
"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.

Parameter String	Data Type	Description
"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
"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 SSR** and **Peak^{TDC} 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.
"PreSamples"	ViInt32	For **AP240/AP235 SSR** and **Peak^{TDC} 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.
"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).
"StartDeltaNegPeak"	ViInt32	Negative excursion needed before searching for negative peak. For **AP101/AP201 Analyzers** ONLY.
"StartDeltaPosPeak"	ViInt32	Positive excursion needed before searching for positive peak. May assume values between 1 and 0xff. For **AP101/AP201 Analyzers** ONLY.
"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.**
"StopDelay "	ViInt32	Stop delay in samples. May assume values between 0 and 1048560 (20971201048560) in steps of of 16 (32) as explained below. The limit is StepSize*(64*1024-1)
"TdcHistogramDepth"	ViInt32	The depth of the histogram for **Peak^{TDC}** mode. 0 means 16-bit accumulation bins. 1 means 32-bit accumulation bins.
"TdcHistogramHorzRes"	ViInt32	The horizontal resolution of the histogram for interpolated peaks in the **Peak^{TDC}** mode. 0 means that each bin corresponds to a sampling interval. ≤4 means that each bin corresponds to ½**n of a sampling interval.
"TdcHistogramIncrement"	ViInt32	The desired increment to be applied for each entry; 1 means increment by 1, for **SimpleTDC Averager** and **Peak^{TDC} Analyzer** modes ONLY. 2 means increment by the ADCvalue – NoiseBase for a **SimpleTDC Averager** and by the ADCvalue for the **Peak^{TDC} Analyzer**

Parameter String	Data Type	Description
"TdcHistogramMode"	ViInt32	The type of histogram for **Peak^{TDC}** mode ONLY. 0 means no histogram. Data only is available for each acquisition. 1 for a histogram.
"TdcHistogramVertRes"	ViInt32	The vertical resolution of the histogram for interpolated peaks when the **TDCHistogramIncrement** is 2 in the **Peak^{TDC}** mode. 0 means that each bin corresponds to a sampling interval. ≤4 means that each bin corresponds to ½**n of a sampling interval.
"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.
"TdcOverlaySegments"	ViInt32	This option controls the horizontal binning of data in the **Peak^{TDC}** histogram mode. 0 means that each segment will be histogrammed independently. 1 means that all segments will be histogrammed on a common time axis.
"TdcProcessType"	ViInt32	The desired processing for **Peak^{TDC}** mode peak finding. May assume 0 = No processing 1 = Standard peak finding (no interpolation) 2 = Interpolated peaks 3 = 8 sample peak regions for data readout 4 = 16 sample peak regions for data readout
"ThresholdEnable"	ViInt32	May assume 0 (no threshold) and 1 (threshold enabled). For **Averagers** ONLY.
"TrigAlways"	ViInt32	May assume 0 (no trigger output) and 1 (trigger output on), in the case of no acquisition.
"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.
"TrigResync"	ViInt32	May assume 0 (no resync), 1 (resync) and 2 (free run)
"ValidDeltaNegPeak"	ViInt32	Positive excursion needed to validate a negative peak. May assume values between 1 and 0xff. For **AP101/AP201** ONLY.
"ValidDeltaPosPeak"	ViInt32	Negative excursion needed to validate a positive peak. May assume values between 1 and 0xff. For **AP101/AP201** 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_configAvgConfigInt32(ID, channelNbr, "DitherRange", dither);
```

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

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

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configAvgConfigInt32(ViSession instrumentID,
                    ViInt32 channelNbr, ViString parameterString,
                    ViInt32 value);
```

LabVIEW Representation

Please use the Acqiris Dx.lvlib: (or Aq Dx) Extended Configure Averager.vi described in **AcqrsD1_configAvgConfig.**

Visual Basic Representation

```
ConfigAvgConfigInt32 (ByVal instrumentID As Long, _
                ByVal channelNbr As Long, _
                ByVal parameterString As String, _
                ByVal value As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configAvgConfigInt32 (ByVal instrumentID As Int32, _
                ByVal channelNbr As Int32, _
                ByVal parameterString As String, _
                ByVal value As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= AqD1_configAvgConfigInt32(instrumentID, channel, parameterString,
                    value)
```

2.3.42 AcqrsD1_configAvgConfigReal64

Purpose

Configures a double 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	ViReal64	Value to set.

Return Value

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

Accepted Parameter Strings

Parameter String	Data Type	Description
"NoiseBase"	ViReal64	Value in Volts of the value to be added in Noise Supressed Averaging. For **Averagers** ONLY.
"StartDeltaPosPeakV"	ViReal64	Positive excursion needed before searching for positive peak. Must be positive. For **Peak^{TDC} mode Analyzers** ONLY.
"Threshold"	ViReal64	Value in Volts of the threshold for **Noise Supressed Averaging** or for **SSR** or **Peak^{TDC}** with **Threshold Gates**.
"ValidDeltaPosPeakV"	ViReal64	Negative excursion needed to validate a positive peak. Must be positive. For **Peak^{TDC}** mode **Analyzers** ONLY.

Discussion

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.

Example

```
long channelNbr = 0;
double threshold = 0.8;
AcqrsD1_configAvgConfigReal64(ID, channelNbr, "DitherRange", double);
```

This function sets the NSA threshold to 0.8 V.

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

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configAvgConfigReal64(ViSession instrumentID,
                    ViInt32 channelNbr, ViString parameterString,
                    ViReal64 value);
```

LabVIEW Representation

Please use the Acqiris Dx.lvlib: (or Aq Dx) Extended Configure Averager.vi described in **AcqrsD1_configAvgConfig**.

Visual Basic Representation

```
ConfigAvgConfigReal64 (ByVal instrumentID As Long, _
                ByVal channelNbr As Long, _
                ByVal parameterString As String, _
                ByVal value As Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configAvgConfigReal64 (ByVal instrumentID As Int32, _
                ByVal channelNbr As Int32, _
                ByVal parameterString As String, _
                ByVal value As Double) As Int32
```

MATLAB MEX Representation

```
[status]= AqD1_configAvgConfigReal64(instrumentID, channel, parameterString,
                    value)
```

2.3.43 AcqrsD1_configChannelCombination

Purpose

Configures how many converters are to be used for which channels. This routine is for use with some DC271-FAMILY instruments, the 10-bit-FAMILY, the AC/SC240, and the AP240/AP235 Signal Analyzer platforms.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
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

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 AS bus, 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configChannelCombination(instrumentID, nbrConvertersPerChannel,
                      usedChannels)

Note: The older form Aq_configChannelCombination is deprecated.
     Please convert to the newer version.
```

2.3.44 AcqrsD1_configControlIO

Purpose

Configures a ControlIO connector. (For DC271-FAMILY/AP-FAMILY/12-bit-FAMILY/10-bit FAMILY and AC/SC 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/ DC271AR/DC122/DC152/DC222/DC252/ DC282) 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. 9 = *Gate* signal for FC option totalize in gate functionality. **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

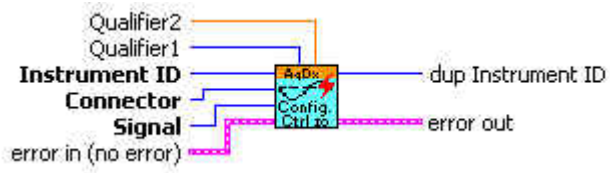
```
AcqrsD1_configControlIO(instrID, 9, 1500, 0, 0.0);
```

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configControlIO(instrumentID, connector, signal, qualifier1,
                      qualifier2)

Note: The older form Aq_configControlIO is deprecated.
     Please convert to the newer version.
```

2.3.45 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

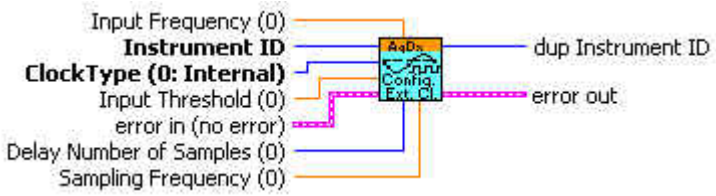
When **clockType** is set to 1 or 4, the parameters of the function **AcqrsD1_configHorizontal** are ignored! Please refer to your product User Manual, for the conditions on the clock signals, and to the **Programmer's Guide** section 3.16**, External Clock**, for the setup parameters and the theory of operation.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configExtClock(instrumentID, clockType, inputThreshold,
                    delayNbrSamples, inputFrequency, sampFrequency)

Note: The older form Aq_configExtClock is deprecated.
     Please convert to the newer version.
```

2.3.46 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
```

MATLAB MEX Representation

```
[status]= AqD1_configFCounter(instrumentID, signalChannel, typeMes,
                     targetValue, apertureTime, reserved, flags)

Note: The older form Aq_configFCounter is deprecated.
     Please convert to the newer version.
```

2.3.47 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

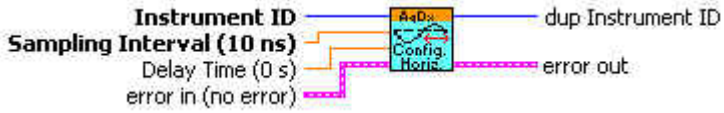
Refer to the **Programmer's Guide** section 3.12, **Trigger Delay and Horizontal Waveform Position**, for a detailed discussion of the value **delayTime**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configHorizontal(instrumentID, sampInterval, delayTime)

Note: The older form Aq_configHorizontal is deprecated.
     Please convert to the newer version.
```

2.3.48 AcqrsD1_configLogicDevice (DEPRECATED)

Purpose

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

NOTE: With the exception of AC and SC Analyzers, this function now needs to be used only by VxWorks users to specify the filePath for FPGA .bit files. Otherwise it should no longer have to be used

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
deviceName	ViChar [ ]	Identifies which device to program For the AC210/AC240 and SC210/SC240 modules this string must be "Block1Dev1". Alternatively it can be "ASBUS::n::Block1Dev1" with n ranging from 0 to the number of modules -1. When clearing the FPGA's, the string must be "Block1DevAll".
filePathName	ViChar [ ]	File path and file name
flags	ViInt32	flags, 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 3 = 0 + use normal search order with AqDrv4.ini file

Return Value

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

Discussion

With flags = 2 in VxWorks systems, the filePathName must point to a directory containing the FPGA configuration files with extension '.bit'

With flags = 0 or 3, 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 flags);
```

LabVIEW Representation

See **Acqrs_configLogicDevice**

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
```

MATLAB MEX Representation

```
[status]= Aq_configLogicDevice(instrumentID, deviceName, filePathName, flags)
```

2.3.49 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configMemory(instrumentID, nbrSamples, nbrSegments)

Note: The older form Aq_configMemory is deprecated.
     Please convert to the newer version.
```

2.3.50 AcqrsD1_configMemoryEx

Purpose

Extended configuration of the memory control parameters of the digitizer including 10-bit-FAMILY & U1071A-FAMILY SAR mode.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
nbrSamplesHi	ViUInt32	Must be set to 0 (reserved for future use)
nbrSamplesLo	ViUInt32	Nominal number of samples to record (per segment!)
nbrSegments	ViInt32	Number of segments to acquire. 1 corresponds to the normal single-trace acquisition mode.
nbrBanks	ViInt32	Number of banks to be used for SAR mode
flags	ViInt32	= 0 default memory use = 1 force use of internal memory (for 10-bit-FAMILY & U1071A-FAMILY digitizers with extended memory options only).

Return Value

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

Discussion

This routine is needed to access the new features of some of the digitizers (U1071A-FAMILY & 10-bit-FAMILY).

The SAR mode should be activated by calling **AcqrsD1_configMode** with the appropriate flags value. The desired number of banks should be set here with the nbrBanks > 1. If the unit has external memory the flags parameter will also have to be set to 1.

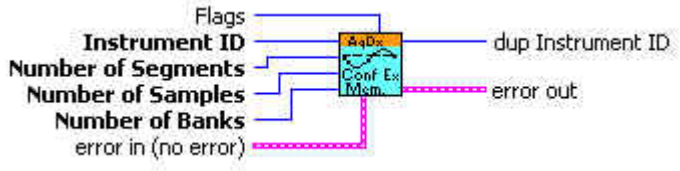
In an instrument equipped with external memory, flags = 1 will force the use of internal memory which give a lower dead time between segments of a sequence acquisition.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_configMemoryEx(ViSession instrumentID, ViUInt32
                    nbrSamplesHi, ViUInt32 nbrSamplesLo, ViInt32 nbrSegments,
                    ViInt32 nbrBanks,
                    ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Configure Extended Memory Settings.vi
```



Visual Basic Representation

```
ConfigMemoryEx (ByVal instrumentID As Long, _
              ByVal nbrSamplesHi As Long, _
              ByVal nbrSamplesLo As Long, _
              ByVal nbrSegments As Long, -
              ByVal nbrBanks As Long, -
              ByVal flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_configMemoryEx (ByVal instrumentID As Int32, _
              ByVal nbrSamplesHi As UInt32, _
              ByVal nbrSamplesLo As UInt32, _
              ByVal nbrSegments As Int32, -
              ByVal nbrBanks As Int32, -
              ByVal flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= AqD1_configMemoryEx(instrumentID, nbrSamplesHi, nbrSamplesLo,
                    nbrSegments, nbrBanks, flags)

Note: The older form Aq_configMemoryEx is deprecated.
     Please convert to the newer version.
```

2.3.51 AcqrsD1_configMode

Purpose

Configures the operational mode of Averagers and Analyzers and certain special Digitizer acquisition modes

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
mode	ViInt32	0 = normal data acquisition 1 = AC/SC stream data to DPU 2 = averaging mode (only in real-time averagers) 3 = buffered data acquisition (only in AP101/AP201 analyzers) 5 = **PeakTDC** mode for Analyzers with this option. 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 10 = SAR 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). AC/SC analyzers support both the normal data acquisition mode (0) *and* the stream data to DPU mode (1).

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 3.18, **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 3.18, **Special Operating Modes**.

- flags = 10: SAR mode. This mode allows simultaneous data acquisition and readout and is available on some models only. **AcqrsD1_configMemoryEx** must be used to set the desired number of banks. When SAR mode is active any external memory present is not available.

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*.

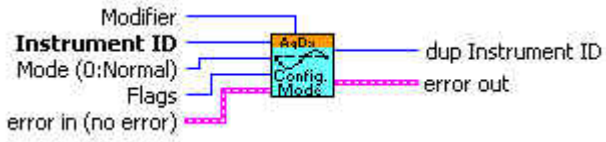
- The functions **AcqrsD1_readData()**: In mode 0, they return the last acquired waveform from the normal acquisition memory, whereas in mode 3, they return data from a memory bank (opposite to what is defined by *flags*).

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configMode(instrumentID, mode, modifier, flags)

Note: The older form Aq_configMode is deprecated.
     Please convert to the newer version.
```

2.3.52 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

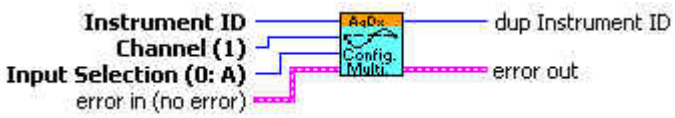
This function is only of use for instruments with an input-multiplexer (i.e. more than 1 input per digitizer, e.g. DP211). On the "normal" instruments with a single input per channel, this function may be ignored.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configMultiInput(instrumentID, channel, input)

Note: The older form Aq_configMultiInput is deprecated.
     Please convert to the newer version.
```

2.3.53 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 in AqGateParameters

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.

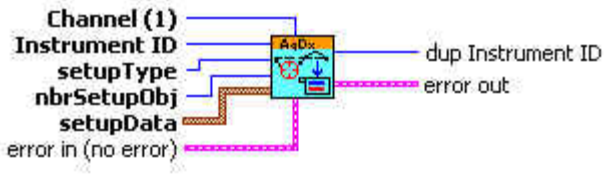
Note: The driver contains a set of 4095(64) default AqGateParameters, defined as { {0,256} {256, 256} {512, 256} {768, 256} ... }.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Configure Setup Array.vi
```



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
```

MATLAB MEX Representation

```
[status]= AqD1_configSetupArray(instrumentID, channel, setupType, nbrSetupObj,
                      setupData)

Note: The older form Aq_configSetupArray is deprecated.
     Please convert to the newer version.
```

2.3.54 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) = 3 OR (10-bit & U1071A-FAMILIES) = 4 NOR (10-bit & U1071A-FAMILIES) = 5 AND (10-bit & U1071A-FAMILIES) = 6 NAND (10-bit & U1071A-FAMILIES)
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* (AS bus operation). See discussion below.
validatePattern	ViInt32	Currently unused, set to "0"
holdType	ViInt32	Currently unused, set to "0"
holdoffTime	ViReal64	Currently unused, set to "0.0"
reserved	ViReal64	Currently unused, set to "0.0"

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 **Acqrs_getInstrumentInfo** function.

For more details on the trigger source pattern in AS bus-connected MultiInstruments, please refer to the **Programmer's Guide** section 3.17.2, **Trigger Source Numbering with AS bus**.

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

The U1071A-FAMILY OR, NOR, AND, and NAND patterns can be implemented as
sourcePattern = 0x800n0001 for Channel 1 +External or
sourcePattern = 0x800n0002 for Channel 2 +External.

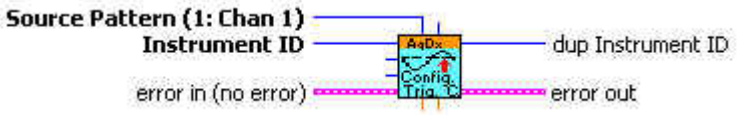
The 10-bit family OR, NOR, AND, and NAND patterns can be implemented as
sourcePattern = *0x8*00n000*f* where ***8*** can be either 8 or 0 and ***f*** can be any value between 0 and f consistent with the number of channels available in a single module.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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 holdoffTime As Double, _
                 ByVal reserved 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 holdoffTime As Double, _
                 ByVal reserved As Double) As Int32
```

MATLAB MEX Representation

```
[status]= AqD1_configTrigClass(instrumentID, trigClass, sourcePattern,
                      validatePattern, holdType, holdoffTime, reserved)

Note: The older form Aq_configTrigClass is deprecated.
     Please convert to the newer version.
```

2.3.55 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 = 5 Spike Stretcher
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 **Acqrs_getInstrumentInfo** function.

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

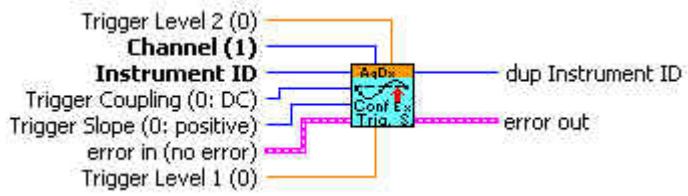
NOTE: Some of the possible states may be unavailable in some digitizers. In particular, the trigCoupling choices of 'DC, 50 Ω' and 'AC, 50 Ω' are only needed for modules that have both 50 Ω and 1 MΩ external input impedance possibilities.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Configure Extended 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
```

MATLAB MEX Representation

```
[status]= AqD1_configTrigSource(instrumentID, channel, trigCoupling,
                    trigSlope, trigLevel1, trigLevel2)

Note: The older form Aq_configTrigSource is deprecated.
     Please convert to the newer version.
```

2.3.56 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

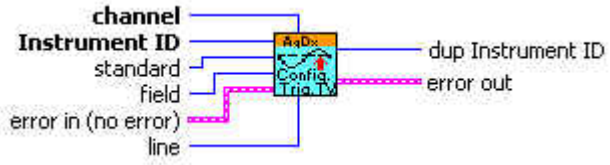
The number of internal (i.e. channel) or external trigger sources of the instrument can be retrieved with the **Acqrs_getInstrumentInfo** function.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configTrigTV(instrumentID, channel, standard, field, line)

Note: The older form Aq_configMemoryEx is deprecated.
     Please convert to the newer version.
```

2.3.57 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.

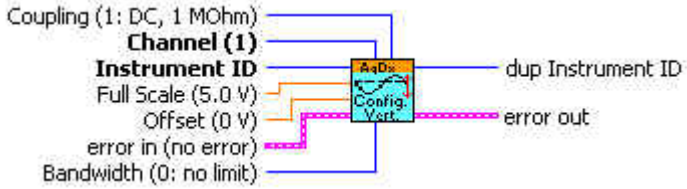
Some instruments have no bandwidth limiting capability. In this case, use **bandwidth** = 0. With **channel** = -1 this function can be used to set the Full Scale Range and the bandwidth limit of the external trigger for the DC271-FAMILY digitizers, the 10-bit-FAMILY, the AC/SC, and the AP240/AP235 signal analyzer platforms. For the case of a 10-bit-FAMILY or DC271-FAMILY *MultiInstrument* using AS bus, the external triggers of the additional modules are numbered –3, -5, … following the principles given in the **Programmer's Guide** section 3.17.2, **Trigger Source Numbering with AS bus**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_configVertical(instrumentID, channel, fullScale, offset,
                      coupling, bandwidth)

Note: The older form Aq_configVertical is deprecated.
     Please convert to the newer version.
```

2.3.58 AcqrsD1_errorMessage

Purpose

Translates an error code into a human readable form. The new function **Acqrs_errorMessage** is to be preferred.

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 (suggested size 512) into which the error-message text is returned

Return Value

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

Discussion

There is no Matlab MEX implementation of this function.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

See **Acqrs_errorMessage**

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
```

2.3.59 AcqrsD1_errorMessageEx

Purpose

Translates an error code into a human readable form and returns associated information. The new function **Acqrs_errorMessage** is to be preferred.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier can be VI_NULL
errorCode	ViStatus	Error code (returned by a function) to be translated
errorMessageSize	ViInt32	Size of the errorMessage string in bytes (suggested size 512)

Output

Name	Type	Description
errorMessage	ViChar [ ]	Pointer to user-allocated string (suggested size 512) into which the error-message text is returned

Return Value

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

Discussion

This function should be called immediately after the return of the error status to ensure that the additional information remains available. For file errors, the returned message will contain the file name and the original 'ansi' error string. This is particularly useful for calls to the following functions:

Acqrs_calibrate	Acqrs_calibrateEx
Acqrs_configLogicDevice	AcqrsD1_configMode
Acqrs_init	Acqrs_InitWithOptions

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_errorMessageEx(ViSession instrumentID, ViStatus
                    errorCode, ViChar errorMessage[], ViInt32
                    errorMessageSize);
```

LabVIEW Representation

See **Acqrs_errorMessage**

Visual Basic Representation

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

Visual Basic .NET Representation

```
AcqrsD1_errorMessageEx (ByVal instrumentID As Int32, _
              ByVal errorCode As Int32, _
              ByVal errorMessage As String,
              ByVal errorMessageSize As Int32) As Int32
```

MATLAB MEX Representation

```
[status errorMessage]= Aq_errorMessage(instrumentID, errorCode)
```

2.3.60 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 ordering the acquisition to stop. 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

See **AcqrsD1_forceTrigEx**

2.3.61 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 ordering the acquisition to stop. 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-AS bus-connected) digitizers (not Averagers or Analyzers).

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status]= AqD1_forceTrigEx(instrumentID, forceTrigType, modifier, flags)

Note: The older form Aq_forceTrigEx is deprecated.
     Please convert to the newer version.
```

2.3.62 AcqrsD1_freeBank

Purpose

Free current bank during SAR acquisitions.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
reserved	ViInt32	Reserved

Return Value

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

Discussion

Calling this function indicates to the driver that the current SAR bank has been read and can be reused for a new acquisition. This call should be made after having read all desired data for the bank.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_freeBank(
                    ViSession instrumentID, ViInt32 reserved);
```

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Free Bank.vi
```

Visual Basic Representation

```
FreeBank (ByVal instrumentID As Long, reserved As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_freeBank (ByVal instrumentID As Int32,
                  ByVal reserved As Int32) As Int32
```

MATLAB MEX Representation

```
[status]= AqD1_freeBank(instrumentID, reserved)
```

2.3.63 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

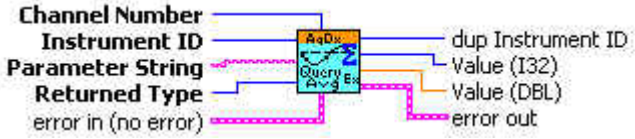
See remarks under **AcqrsD1_configAvgConfig**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Extended Averager Settings.vi
      This Vi returns the value as either I32 or DBL. Connect the desired
type.
```



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
```

MATLAB MEX Representation

Please use the MEX representation associated with **AcqrsD1_getAvgConfigInt32** or **AcqrsD1_getAvgConfigReal64**

Note: The older form Aq_getAvgConfig is deprecated.
Please convert to the newer version.

2.3.64 AcqrsD1_getAvgConfigInt32

Purpose

Returns a long 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	ViInt32 *addr	Requested information value.

Return Value

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

Discussion

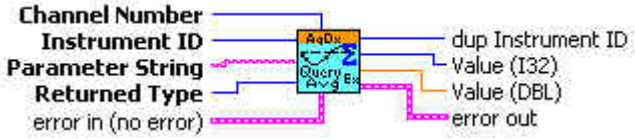
See remarks under **AcqrsD1_configAvgConfig**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getAvgConfigInt32(ViSession instrumentID,
                     ViInt32 channelNbr, ViString parameterString,
                     ViInt32 *value);
```

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Extended Averager Settings.vi
      This Vi returns the value as either I32 or DBL. Connect the desired
type.
```



Visual Basic Representation

```
GetAvgConfigInt32 (ByVal instrumentID As Long, _
              ByVal channelNbr As Long, _
              ByVal parameterString As String, _
              value as Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getAvgConfigInt32 (ByVal instrumentID As Int32, _
              ByVal channelNbr As Int32, _
              ByVal parameterString As String, _
              ByRef value as Int32) As Int32
```

MATLAB MEX Representation

```
[status value]= AqD1_getAvgConfigInt32(instrumentID, channel, parameterString)
```

2.3.65 AcqrsD1_getAvgConfigReal64

Purpose

Returns a double 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	ViReal64 *	Requested information value.

Return Value

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

Discussion

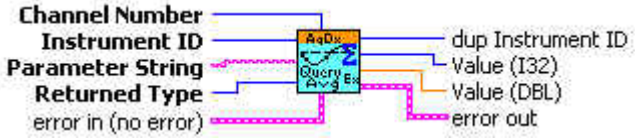
See remarks under **AcqrsD1_configAvgConfig**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getAvgConfigReal64(ViSession instrumentID,
                     ViInt32 channelNbr, ViString parameterString,
                     ViReal64 *value);
```

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Extended Averager Settings.vi
      This Vi returns the value as either I32 or DBL. Connect the desired
type.
```



Visual Basic Representation

```
GetAvgConfigReal64 (ByVal instrumentID As Long, _
              ByVal channelNbr As Long, _
              ByVal parameterString As String, _
              value as Double) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getAvgConfigReal64 (ByVal instrumentID As Int32, _
              ByVal channelNbr As Int32, _
              ByVal parameterString As String, _
              ByRef value as Double) As Int32
```

MATLAB MEX Representation

```
[status value]= AqD1_getAvgConfigReal64(instrumentID, channel,
                     parameterString)
```

2.3.66 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

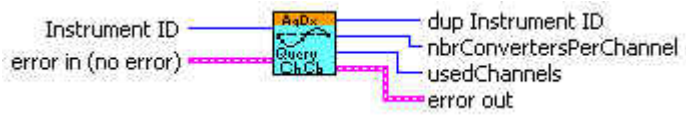
See remarks under **AcqrsD1_configChannelCombination**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status nbrConvertersPerChannel usedChannels]=
                      AqD1_getChannelCombination(instrumentID)

Note: The older form Aq_getChannelCombination is deprecated.
     Please convert to the newer version.
```

2.3.67 AcqrsD1_getControlIO

Purpose

Returns the configuration of a ControlIO connector. (For DC271-FAMILY/10-bit-FAMILY/AP-FAMILY/12-bit-FAMILY and AC/SC Analyzers 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)

Output

Name	Type	Description
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.

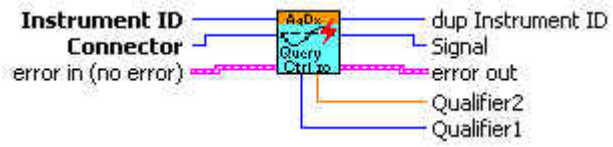
The DC271-FAMILY, 10-bit-FAMILY, AP-FAMILY, 12-bit-FAMILY, and AC/SC cards support the connectors 1 (front panel I/O A MMCX coax), 2 (front panel I/O B MMCX coax) and 9 (front panel Trig Out MMCX coax).

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status signal qualifier1 qualifier2]= AqD1_getControlIO(instrumentID,
                    connector)

Note: The older form Aq_getControlIO is deprecated.
     Please convert to the newer version.
```

2.3.68 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

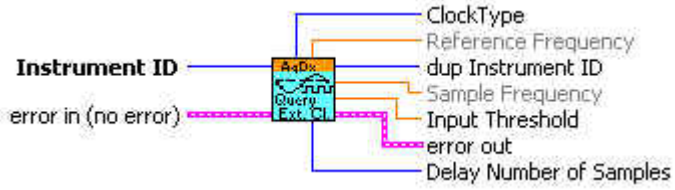
See remarks under **AcqrsD1_configExtClock**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status clockType inputThreshold delayNbrSamples inputFrequency
                    sampFrequency]= AqD1_getExtClock(instrumentID)

Note: The older form Aq_getExtClock is deprecated.
     Please convert to the newer version.
```

2.3.69 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status signalChannel typeMes targetValue apertureTime reserved flags]=
                     AqD1_getFCounter(instrumentID)

Note: The older form Aq_getFCounter is deprecated.
     Please convert to the newer version.
```

2.3.70 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status sampInterval delayTime] = AqD1_getHorizontal(instrumentID)

Note: The older form Aq_getHorizontal is deprecated.
     Please convert to the newer version.
```

2.3.71 AcqrsD1_getInstrumentData (DEPRECATED)

Purpose

Returns some basic data about a specified digitizer. See **Acqrs_getInstrumentData**.

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

Please refer to **Acqrs_getInstrumentData**

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
```

MATLAB MEX Representation

```
[status name serialNbr busNbr slotNbr]= Aq_getInstrumentData(instrumentID)
```

2.3.72 AcqrsD1_getInstrumentInfo (DEPRECATED)

Purpose

Returns general information about a specified digitizer. See **Acqrs_getInstrumentInfo**.

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	Returns 1 if the *m*'th module of a multi-instrument is the master, 0 otherwise. *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.
`"LogDevDataLinks"`	ViInt32	Number of available data links for a streaming analyzer
`"LOGDEVHDRBLOCK`*m*`DEV`*n*`S`*string*`"`	ViChar[ ]	Returns information about FPGA firmware loaded. See comments below.
`"MaxSamplesPerChannel"`	ViInt32	Maximum 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 (not connected through AS bus) return 1.
`"Options"`	ViChar[ ]	List of options, separated by ‘,’, installed in this instrument.

Parameter String	Returned Type	Description
"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
"SSRTimeStamp"	ViReal64	Current value of time stamp for Analyzers in SSR mode.
"TbSegmentPad"	ViInt32	Returns the additional array space (in samples) per segment needed for the image read of **AcqrsD1_readData**. It concerns the current data available, as opposed to any future acquisition with different conditions.
"Temperature *m*"	ViInt32	Temperature in degrees Centigrade (^{o}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".

The case of "SSRTimeStamp" should only be used when data is readable. In other words, it should only be used between the moment at which the processing is done and the moment when AcqrsD1_processData is called to enable the subsequent bank switch.

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

Please refer to **Acqrs_getInstrumentInfo**.

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
```

MATLAB MEX Representation

```
[status infoValue] = Aq_getInstrumentInfo(instrumentID, parameterString,
                    dataTypeString)
```

Allowed values of dataTypeString are 'integer', 'double', or 'string'.

2.3.73 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status nbrSamples nbrSegments] = AqD1_getMemory(instrumentID)

Note: The older form Aq_getMemory is deprecated.
     Please convert to the newer version.
```

2.3.74 AcqrsD1_getMemoryEx

Purpose

Returns the current extended memory control parameters of the digitizer.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
nbrSamplesHi	ViUInt32	Will be set to 0 (reserved for future use)
nbrSamplesLo	ViUInt32	Nominal number of samples to record (per segment!)
nbrSegments	ViInt32	Number of segments to acquire. 1 corresponds to the normal single-trace acquisition mode.
nbrBanks	ViInt32	Number of banks to be used for 10-bit-FAMILY & U1071A-FAMILY SAR mode
flags	ViInt32	= 0 default memory use = 1 force use of internal memory (for 10-bit-FAMILY & U1071A-FAMILY digitizers with extended memory options only).

Return Value

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

Discussion

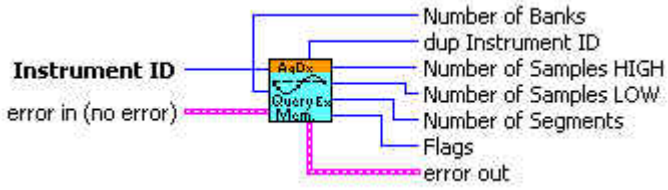
See remarks under **AcqrsD1_configMemoryEx**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_getMemoryEx(ViSession instrumentID,
                    ViUInt32* nbrSamplesHi, ViUInt32* nbrSamplesLo, ViInt32*
                    nbrSegments, ViInt32* nbrBanks,
                    ViInt32* flags);
```

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Extended Memory Settings.vi
```



Visual Basic Representation

```
GetMemoryEx (ByVal instrumentID As Long, _
              nbrSamplesHi As Long, _
              nbrSamplesLo As Long, _
              nbrSegments As Long, -
              nbrBanks As Long, -
              flags As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_getMemoryEx (ByVal instrumentID As Int32, _
              ByRef nbrSamplesHi As UInt32, _
              ByRef nbrSamplesLo As UInt32, _
              ByRef nbrSegments As Int32, -
              ByRef nbrBanks As Int32, -
              ByRef flags As Int32) As Int32
```

MATLAB MEX Representation

```
[status nbrSamplesHi nbrSamplesLo nbrSegments nbrBanks flags]=
                    AqD1_getMemoryEx(instrumentID)

Note: The older form Aq_getMemoryEx is deprecated.
     Please convert to the newer version.
```

2.3.75 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.

Discussion

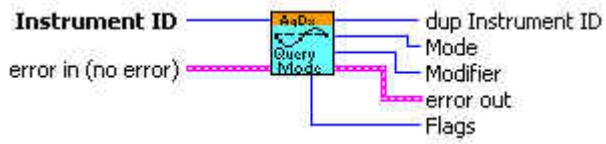
See remarks under **AcqrsD1_configMode.**

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status mode modifiers flags] = AqD1_getMode(instrumentID)

Note: The older form Aq_getMode is deprecated.
     Please convert to the newer version.
```

2.3.76 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status input] = AqD1_getMultiInput(instrumentID, channel)

Note: The older form Aq_getMultiInput is deprecated.
     Please convert to the newer version.
```

2.3.77 AcqrsD1_getNbrChannels (DEPRECATED)

Purpose

Returns the number of channels on the specified module. See **Acqrs_getNbrChannels**.

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

Please refer to **Acqrs_getNbrChannels**

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
```

MATLAB MEX Representation

```
[status nbrChannels] = Aq_getNbrChannels(instrumentID)
```

2.3.78 AcqrsD1_getNbrPhysicalInstruments (DEPRECATED)

Purpose

Returns the number of physical Acqiris modules found on the computer. See **Acqrs_getNbrInstruments**.

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.

Discussion

In the case of multiple processes accessing the Agilent Acqiris instruments, this function will return the number of currently available instruments. If an instrument has already been initialized in another process, it will not be available unless it has been suspended via a call to Acqrs_suspendControl.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

Please refer to **Acqrs_getNbrInstruments.**

Visual Basic Representation

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

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status nbrInstrument]= Aq_getNbrPhysicalInstruments()
```

2.3.79 AcqrsD1_getSetupArray

Purpose

Returns an array of configuration parameters. 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	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	ViInt32	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.

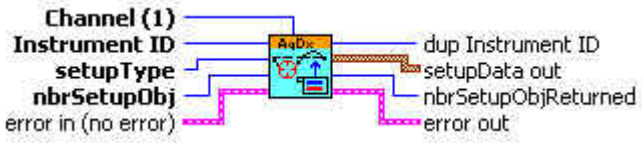
Note: The driver contains a set of 64 default AqGateParameters, defined as { {0,256} {256, 256} {512, 256} {768, 256} ... }.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Setup Array.vi
```



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
```

MATLAB MEX Representation

```
[status setupData nbrSetupObjReturned] = AqD1_getSetupArray(instrumentID,
                      channel, setupType, nbrSetupObj)

Note: The older form Aq_getSetupArray is deprecated.
     Please convert to the newer version.
```

2.3.80 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 (12-bit-FAMILY External only) = 3 OR (10-bit & `U1071A-FAMILIES`) = 4 NOR (10-bit & `U1071A-FAMILIES`) = 5 AND (10-bit & `U1071A-FAMILIES`) = 6 NAND (10-bit & `U1071A-FAMILIES`)
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* (AS bus operation). See discussion below.
validatePattern	ViInt32	Currently returns "0"
holdType	ViInt32	Currently returns "0"
holdoffTime	ViReal64	Currently returns "0"
reserved	ViReal64	Currently returns "0"

Return Value

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

Discussion

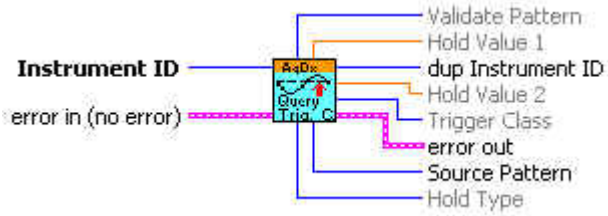
See remarks under **AcqrsD1_configTrigClass**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Trigger Class.vi
```



Visual Basic Representation

```
GetTrigClass (ByVal instrumentID As Long, _
              trigClass As Long, _
              sourcePattern As Long, _
              validatePattern As Long, _
              holdType As Long, _
              holdoffTime As Double, _
              reserved 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 holdoffTime As Double, _
              ByRef reserved As Double) As Int32
```

MATLAB MEX Representation

```
[status trigClass sourcePattern validatePattern holdType holdoffTime reserved]
                    = AqD1_getTrigClass(instrumentID)

Note: The older form Aq_getTrigClass is deprecated.
     Please convert to the newer version.
```

2.3.81 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 = 5 Spike Stretcher
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

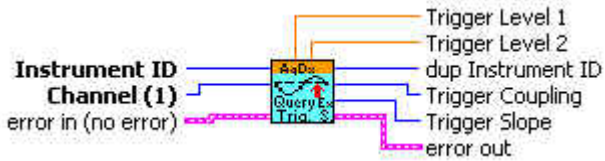
See remarks under **AcqrsD1_configTrigSource**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) Query Extended 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
```

MATLAB MEX Representation

```
[status trigCoupling trigSlope trigLevel1 trigLevel2] =
                    AqD1_getTrigSource(instrumentID, channel)

Note: The older form Aq_getTrigSource is deprecated.
     Please convert to the newer version.
```

2.3.82 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

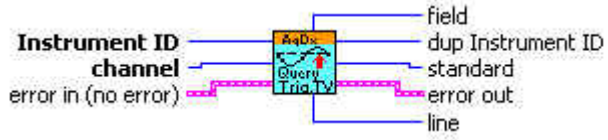
See discussion under **AcqrsD1_configTrigTV**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status standard field line] = AqD1_getTrigTV(instrumentID, channel)

Note: The older form Aq_getTrigTV is deprecated.
     Please convert to the newer version.
```

2.3.83 AcqrsD1_getVersion (DEPRECATED)

Purpose

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

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

Please refer to **`Acqrs_getVersion`**.

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
```

MATLAB MEX Representation

```
[status version] = Aq_getVersion(instrumentID, versionItem)
```

2.3.84 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

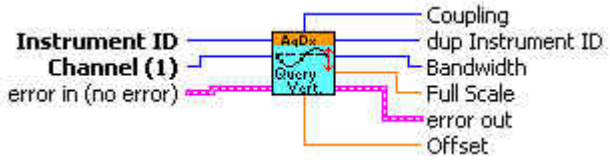
See remarks under **AcqrsD1_configVertical**.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status fullScale offset coupling bandwidth] = AqD1_getVertical(instrumentID,
                     channel)

Note: The older form Aq_getVertical is deprecated.
     Please convert to the newer version.
```

2.3.85 AcqrsD1_init (DEPRECATED)

Purpose

Initializes an instrument. See **Acqrs_init**.

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

Please refer to **Acqrs_init**.

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
```

MATLAB MEX Representation

```
[status instrumentID] = Aq_init(instrumentID, IDQuery, resetDevice)
```

2.3.86 AcqrsD1_InitWithOptions (DEPRECATED)

Purpose

Initializes an instrument with options. See **Acqrs_InitWithOptions**.

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

Please refer to **Acqrs_InitWithOptions**.

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
```

MATLAB MEX Representation

```
[status instrumentID]= Aq_initWithOptions(resourceName, IDQuery, resetDevice,
                    optionsString)
```

2.3.87 AcqrsD1_logicDeviceIO (DEPRECATED)

Purpose

Reads/writes a number of 32-bit data values from/to a user-defined register in on-board logic devices, such as user-programmable FPGAs. It is useful for AC/SC Analyzers only. See **Acqrs_logicDeviceIO**.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
deviceName	ViChar [ ]	Identifies which device to read from or write to. In the AC210/240 and the SC210/240, this string must be "Block1Dev1"
registerID	ViInt32	Register Number, can typically assume 0 to 127
nbrValues	ViInt32	Number of data values to read
dataArray	ViInt32 [ ]	User-supplied array of data values
readWrite	ViInt32	Direction 0 = read from device, 1 = write to device
flags	ViInt32	Currently unused, set to "0"

Return Value

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

Discussion

This function is only useful if the user programmed the on-board logic device (FPGA).

Typically, *nbrValues* is set to 1, but it may be larger if the logic device supports internal address auto-incrementation. The following example reads the (32-bit) contents of register 5 to *reg5Value*:

```
ViStatus status =
     AcqrsD1_logicDeviceIO(ID, "Block1Dev1", 5, 1, &reg5Value, 0, 0);
```

Note that *dataArray* must always be supplied as an address, even when writing a single value.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_logicDeviceIO(ViSession instrumentID,
                    ViChar deviceName[], ViInt32 registerID,
                    ViInt32 nbrValues,   ViInt32 dataArray[],        ViInt32
                    readWrite,   ViInt32 flags);
```

LabVIEW Representation

Please refer to **Acqrs_logicDeviceIO**.

Visual Basic Representation

```
LogicDeviceIO      (ByVal instrumentID As Long, _
                    ByVal deviceName As String, _
                    ByVal registerID As Long, _
                    ByVal nbrValues As Long, _
                    dataArray As Long, _
                    ByVal readWrite As Long, _
                    ByVal modifier As Long) As Long
```

Visual Basic .NET Representation

```
AcqrsD1_logicDeviceIO (ByVal instrumentID As Int32, _
                       ByVal deviceName As String, _
                       ByVal registerID As Int32, _
                       ByVal nbrValues As Int32, _
                       ByRef dataArray As Int32, _
                       ByVal readWrite As Int32, _
                       ByVal modifier As Int32) As Int32
```

MATLAB MEX Representation

```
[status] = Aq_logicDeviceIO(instrumentID, deviceName, registerID, nbrValues,
                    dataArray, readWrite, modifier)
```

2.3.88 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 AS bus.

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 AS bus.

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 **Acqrs_init** or **Acqrs_InitWithOptions** to retrieve the **instrumentID** of the (multi)digitizers.

In the case of multiple processes accessing the Agilent Acqiris instruments, this function will return the number of currently available instruments. If an instrument has already been initialized in another process, it will not be available unless it has been suspended via a call to Acqrs_suspendControl.

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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status nbrInstruments] = AqD1_multiInstrAutoDefine(optionsString)

Note: The older form Aq_multiInstrAutoDefine is deprecated.
     Please convert to the newer version.
```

2.3.89 AcqrsD1_multiInstrDefine

Purpose

This function defines the combination of a number of digitizers connected by AS bus into a single *MultiInstrument*. It is not applicable to AS bus 2 modules.

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 3.2.8, **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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status instrumentID] = AqD1_multiInstrDefine(instrumentList, nbrInstruments,
                    masterID)

Note: The older form Aq_multiInstrDefine is deprecated.
     Please convert to the newer version.
```

2.3.90 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**s 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status] = AqD1_multiInstrUndefineAll(optionsString)

Note: The older form Aq_multiInstrUndefineAll is deprecated.
     Please convert to the newer version.
```

2.3.91 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

```
Acqiris Dx.lvlib: (or Aq Dx) Query Process Done.vi
```

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
```

MATLAB MEX Representation

```
[status done] = AqD1_procDone(instrumentID)

Note: The older form Aq_procDone is deprecated.
     Please convert to the newer version.
```

2.3.92 AcqrsD1_processData

Purpose

Starts on-board data processing on acquired data in the current bank as soon as the current acquisition 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 (or other Analyzers) and for **AP101/AP201 ONLY** 1 = gated peak detection, extrema mode 2 = gated peak detection, hysteresis mode 3 = interpolated peaks, extrema mode 4 = interpolated peaks, hysteresis mode And for **PeakTDC Analyzers** 0 = respect the settings done with AcqrsD1_configAvgConfig 1 = gated peak detection with hystersis 2 = gated and interpolated peak detection with hysteresis 3 = gated peak detection with 8-point peak region 4 = gated peak detection with 16-point peak region
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

```
Acqiris Dx.lvlib: (or Aq Dx) Process Data.vi
```

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
```

MATLAB MEX Representation

```
[status] = AqD1_processData(instrumentID, processType, flags)

Note: The older form Aq_processData is deprecated.
     Please convert to the newer version.
```

2.3.93 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** or **Peak^{TDC}**) 5 = short averaged waveform (from an **Averager**) 6 = shifted short averaged waveform (from an **Averager**) 7 = gated data from an **SSR** or **Peak^{TDC} Analyzer** 9 = **Peak^{TDC}** Histogram readout from an **Analyzer** 10 = **Peak^{TDC}** Peak region readout from an **Analyzer** 11 = raw sequence waveform read
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	As used for DIGITIZERS Bit 0 LSB: If set the first data point is at a fixed number of points with respect to the resynchronized trigger, otherwise it is before delayTime after the Trigger Bit 1: If set the lookup table (if any) will not be used to translate the data, otherwise it will be. Bit 3: If set the memory image will be transferred in an image read but no segment re-ordering will be done, otherwise it will be. For Averagers if Bit 2 is set the accumulated data will not be reset after being read, otherwise it will be. AcqirisDataTypes.h contains AqReadDataFlags an enum which encodes the above values.
reserved	ViInt32	Reserved for future use, set to 0.
reserved2	ViReal64	Reserved for future use, set to 0.
reserved3	ViReal64	Reserved for future use, set to 0.

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 = 2,5,6) 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.
flags	ViUInt32	The lowest four bits contain the hardware marker values. The correspondence is Bit 0 (LSB) = P1, Bit 1 = P2 Bit 2 = I/O A Bit 3 = I/O B The marker is set at the last trigger, in the first round of the acquisition of the segment.
reserved	ViInt32	Reserved for future use

Segment Descriptor for Raw Sequence Waveforms (readMode = 11) in AqSegmentDescriptorSeqRaw

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.
indexFirstPoint	ViUInt32	Pointer to first sample of this segment
actualSegmentSize	ViUInt32	Actual segment size, for the size of the circular buffer
reserved	ViInt32	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. This value is only returned for **SSR** and **Peak^{TDC} Analyzers.**
reserved2	ViInt32	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**. This file also contains **enum** definitions for the allowed values of the members of the **AqReadParameters** structure.

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

Remember to set all values of the **AqReadParameters** structure, including the reserved values.

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

	0 = standard	1 = image	2 = averaged	3 = gated	4 = peaks
0 = Int8	8,10	8,10	-	APX01	-
1 = Int16	10,12	10,12	-	-	-
2 = Int32	-	-	X	-	Peak^{TDC}
3 = Real64	X	X	X	-	APX01

	5 = short averaged	6 = shifted short averaged	7 = SSR	9 = Histogram	10 = peak region	11 = sequence raw
0 = Int8	-	-	X			8,10
1 = Int16	X	X	-	Peak^{TDC}		10,12
2 = Int32	-	-	-	Peak^{TDC}	Peak^{TDC}	
3 = Real64	X	X	-			

In this table

‘X’ means that the functionality is available depending on the option but independent of the model,

'8' means that the functionality is available for 8-bit Digitizers and AP units in the digitizer mode,

'10' means that it is available for the 10-bit Digitizers,

'12' means that it is available for the 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.

10-bit digitizers generate 12-bit data which can be transferred in either of 2 ways

- 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
- 1 byte with the 8-bit data of the most significant bits of the ADC value. Here the lowest two bits will be lost (truncated). The advantage is that the amount of data to be transferred has been cut by a factor of 2.

Real64 readout of 10-bit digitizers is based on 16-bit transfer of the data,

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.

For **readMode = 0** and **dataType ≤ 1, indexFirstPoint** must be used for the correct identification of the first data point in the **dataArray**.

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 3.10.5, **Reading an Averaged Waveform from an Averager** and 3.10.6, **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 max(40,8*nbrSamplesInSeg) for 8-bit digitizers and max(88,8*nbrSamplesInSeg) for 10-bit and 12-bit digitizers.
- with readMode = 1 or readMode = 11 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 3.10.2, **Reading Sequences of Waveforms.**

- For averagers
  - with readMode = 0,1 cannot be used. If the AcqrsD1_configMode mode is set to 0 (normal data acquisition) please use the digitizer rules above
  - with 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 cannot be used. If the AcqrsD1_configMode mode is set to 0 (normal data acquisition) please use the digitizer rules above
  - readMode = 2 cannot be used
  - with readMode = 3 the array size must be at least the sum of all gate lengths.
  - with readMode = 4 in the APx01 analyzers the array size must be 4*sizeof(double) * number of gates
  - with readMode = 4 in the **Peak**^{TDC} analyzers the array size must be 8 * number of peaks
  - with readMode = 7 in the **Peak**^{TDC} or SSR analyzers the array size must be nbrSegments * (16 + nbrSamplesInSeg) for the simple case of all the data in a single gate. For other cases please see the **Programmer's Guide** section 3.10.7, **Reading SSR Analyzer Waveforms**, for a detailed explanation.
  - with readMode = 9 the array size must be at least
    - 2**HistoRes*nbrSamplesInSeg* nbrSegments*Size_of_dataType if a segmented histogram is used and

    where
    - HistoRes is the value used in the call to **Acqrs_configAvgConfig** with "TdcHistogramRes"
    - NbrSegments is either 1 or the number of segments if the value used in the call to **Acqrs_configAvgConfig** with "TdcHistogramMode" is 1
    - Size_dataType = 2*(1+HistoDepth), where HistoDepth is the value used in the call to **Acqrs_configAvgConfig** with "TdcHistogramDepth"
  - 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.

If used 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**. If not used a Null pointer can be passed to the function. There are no segment descriptors for readMode = 4, 7, 9, and 10.

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 (usually the start-time of the acquisition except for the 10-bit digitizers), which is intended for the computation of time differences between segments of a Sequence acquisition. Please refer to the **Programmer's Guide** section 3.15, **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 3.12, **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

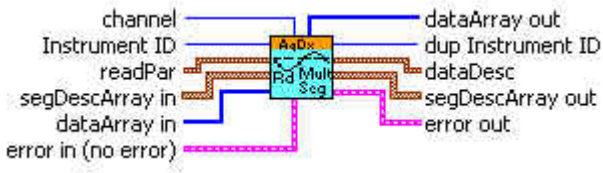
```
ViStatus status = AcqrsD1_readData(ViSession instrumentID,
                    ViInt32 channel, AqReadParameters* readPar,
                    ViAddr dataArray, AqDataDescriptor* descriptor, ViAddr
                    segDesc);
```

LabVIEW Representations

`Acqiris Dx.lvlib: (or Aq Dx) Read Multi-Segments.vi`

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

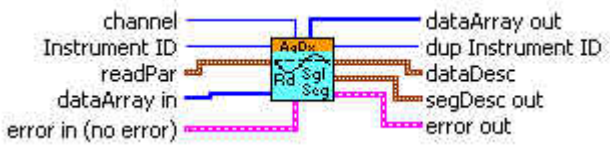
It is meant for the readout of multiple segments with readMode = 1.



`Acqiris Dx.lvlib: (or Aq Dx) Read Single Segment.vi`

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

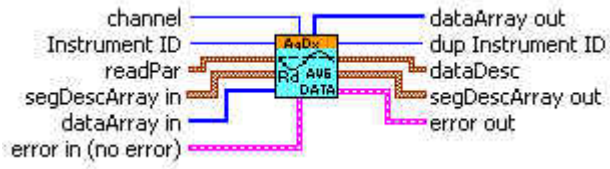
It is meant for the readout of a single segment with readMode = 0.



Acqiris Dx.lvlib: (or Aq Dx) Read Averager Data.vi

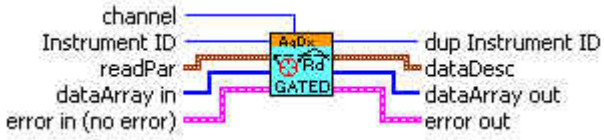
This Vi is polymorphic, the sample data is returned in an array of type I32 or DBL

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



Acqiris Dx.lvlib: (or Aq Dx) Read Gated Data.vi

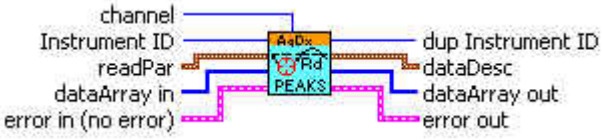
It is meant for the readout of an analyzer with readMode = 3.



Acqiris Dx.lvlib: (or Aq Dx) Read Peaks Data.vi

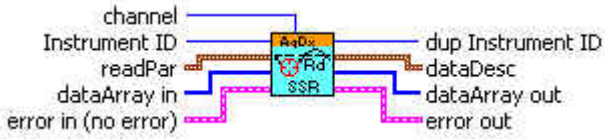
This Vi is polymorphic, the sample data is returned in an array of type I32 or DBL

It is meant for the readout of an analyzer with readMode = 4.



Acqiris Dx.lvlib: (or Aq Dx) Read SSR Data.vi

It is meant for the readout of an analyzer with readMode = 7.



Acqiris Dx.lvlib: (or Aq Dx) Read Histogram Data.vi

This Vi is polymorphic, the sample data is returned in an array of type I16 or I32

It is meant for the readout of an **Peak**[TDC] analyzer with readMode = 4.

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
```

Note: For readPar.readMode = 1 you must use dataType=3;

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
```

Note: For readPar.readMode = 1 you must use dataType=3;

```
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
```

MATLAB MEX Representation

```
[status dataDesc segDescArray dataArray] = AqD1_readData(instrumentID,
                    channel, readPar)

Note: The older form Aq_readData is deprecated.
     Please convert to the newer version.
```

2.3.94 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status result] = AqD1_readFCounter(instrumentID)

Note: The older form Aq_readFCounter is deprecated.
     Please convert to the newer version.
```

2.3.95 AcqrsD1_reportNbrAcquiredSegments

Purpose

Returns the number of segments already acquired for a digitizer. For averagers (but not AP100 or AP200) it will give the number of triggers already accepted for the current acquisition. In the case of analyzers it will return the value 1 at the end of the acquisition and is therefore not of much use.

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

⚠ ***NOTE: For a digitizer, calling this function while an acquisition is active, in order to follow the progress of a Sequence acquisition, is dangerous and must be avoided.***

As needed the result should be interpreted as a ViUInt32.

LabWindowsCVI/Visual C++ Representation

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

LabVIEW Representation

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status nbrSegments] = Aqd1_reportNbrAcquiredSegments(instrumentID)

Note: The older form Aq_reportNbrAcquiredSegments is deprecated.
     Please convert to the newer version.
```

2.3.96 AcqrsD1_reset (DEPRECATED)

Purpose

Resets an instrument. See **Acqrs_reset**

.

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

Please refer to **Acqrs_reset**.

Visual Basic Representation

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

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status] = Aq_reset(instrumentID)
```

2.3.97 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

```
Please refer to Acqrs_resetMemory.
```

Visual Basic Representation

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

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status] = AqD1_resetDigitizerMemory(instrumentID)

Note: The older form Aq_resetDigitizerMemory is deprecated.
     Please convert to the newer version or Aq_resetMemory.
```

2.3.98 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 **`Acqrs_getInstrumentInfo`** (…, "DelayOffset", ..) before power-off If not known, use the value –20.0e-9
delayScale	ViReal64	Global delay scale, should be retrieved with **`Acqrs_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 3.19, **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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status] = AqD1_restoreInternalRegisters(instrumentID, delayOffset,
                      delayScale)

Note: The older form Aq_restoreInternalRegisters is deprecated.
     Please convert to the newer version.
```

2.3.99 AcqrsD1_setAttributeString (DEPRECATED)

Purpose

Sets an attribute with a string value (for use in SC Streaming Analyzers ONLY).

See **Acqrs_setAttributeString**

.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1...Nchan
name	ViConstString	ASCII string that specifies options “odlTxBitRate” is currently the only one used
value	ViConstString	For “odlTxBitRate” can have values like “2.5G”,”2.125G”, or “1.0625G”

Return Value

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

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsD1_setAttributeString(ViSession instrumentID,
                    ViInt32 channel, ViConstString name,
                    ViConstString value);
```

LabVIEW Representation

Please refer to **Acqrs_setAttributeString**.

Visual Basic Representation

Please refer to **Acqrs_resumeControl**.

Visual Basic .NET Representation

Please refer to **Acqrs_resumeControl**.

MATLAB MEX Representation

Please refer to **Acqrs_resumeControl**.

2.3.100 AcqrsD1_setLEDColor (DEPRECATED)

Purpose

Sets the front panel LED to the desired color. See **Acqrs_setLEDColor**.

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

Please refer to **Acqrs_setLEDColor**.

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
```

MATLAB MEX Representation

```
[status] = Aq_setLEDColor(instrumentID, color)
```

2.3.101 AcqrsD1_setSimulationOptions (DEPRECATED)

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". See **Acqrs_setSimulationOptions**

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 3.2.10, **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 Acqiris Bx.lvlib: (or Aq Bx) 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
```

MATLAB MEX Representation

```
[status] = Aq_setSimulationOptions(simOptionsString)
```

2.3.102 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status] = AqD1_stopAcquisition(instrumentID)

Note: The older form Aq_stopAcquisition is deprecated.
     Please convert to the newer version.
```

2.3.103 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

```
Acqiris Dx.lvlib: (or Aq Dx) Stop Processing.vi
```

Visual Basic Representation

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

Visual Basic .NET Representation

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

MATLAB MEX Representation

```
[status] = AqD1_stopProcessing(instrumentID)

Note: The older form Aq_stopProcessing is deprecated.
     Please convert to the newer version.
```

2.3.104 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 comes first. 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

```
Acqiris Dx.lvlib: (or Aq Dx) 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
```

MATLAB MEX Representation

```
[status] = AqD1_waitForEndOfAcquisition(instrumentID, timeOut)

Note: The older form Aq_waitForEndOfAcquisition is deprecated.
     Please convert to the newer version.
```

2.3.105 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 comes first. 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

```
Acqiris Dx.lvlib: (or Aq Dx) Wait For End Of Processing.vi
```

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
```

MATLAB MEX Representation

```
[status] = AqD1_waitForEndOfProcessing(instrumentID, timeOut)

Note: The older form Aq_waitForEndOfProcessing is deprecated.
     Please convert to the newer version.
```

2.3.106 AcqrsT3_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 = AcqrsT3_acqDone(ViSession instrumentID,
                      ViBoolean* done);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Query Acquisition Status.vi
```

MATLAB MEX Representation

```
[status done]= AqT3_acqDone(instrumentID)
```

2.3.107 AcqrsT3_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.

Discussion

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_acquire(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Start Acquisition.vi
```

MATLAB MEX Representation

```
[status done]= AqT3_acquire(instrumentID)
```

2.3.108 AcqrsT3_configAcqConditions

Purpose

Configures parameters affecting the entire acquisition.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
timeout	ViReal64	Timeout in seconds
flags	ViInt32	The LSB (bit 0) = 0 start timeout counter on Arm = 1 start timeout counter on first Common hit
reserved	ViInt32	Currently unused, set to "0"

Return Value

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

Discussion

The timeout value of 0.0 means no timeout.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_configAcqConditions(ViSession instrumentID,
         ViReal64 timeout, ViInt32 flags, ViInt32 reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Configure Acquisition Conditions.vi
```

MATLAB MEX Representation

```
[status ]= AqT3_configAcqConditions(instrumentID, timeout, flags, reserved)
```

2.3.109 AcqrsT3_configChannel

Purpose

Configures parameters for defining timing events on each channel.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1…Nchan or −1 for the common channel −2 for the veto
mode	ViInt32	The LSB (bit 0) = 0 positive slope = 1 negative slope Bit 1 = 0 normal events = 1 pulse events with pulse type defined by the LSB (TC890 ONLY) The MSB (bit31) = 0 active channel = 1 inactive channel
level	ViReal64	Threshold value in Volts.
reserved	ViInt32	Currently unused, set to "0"

Return Value

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

Discussion

Nchan can be found from a call to **Acqrs_getNbrChannels.**

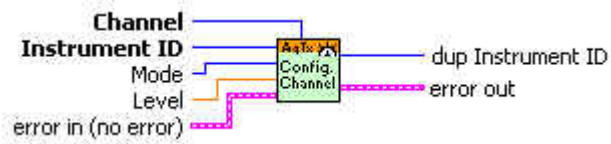
The common channel cannot be inactivated.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_configChannel(ViSession instrumentID, ViInt32
                    channel, ViInt32 mode, ViReal64 level, ViInt32 reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Configure Channel.vi
```



MATLAB MEX Representation

```
[status]= AqT3_configChannel(instrumentID, channel, mode, level, reserved)
```

2.3.110 AcqrsT3_configControlIO

Purpose

Configures the auxiliary I/O connectors.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
connector	ViInt32	Connector Number 1 = Front Panel I/O Aux 1 2 = Front Panel I/O Aux 2 13 = Front Panel Veto Input
signal	ViInt32	See below
qualifier1	ViInt32	If the LSB (bit0) is set to 1 and the connector is being used for an input signal, forces 50 Ohm termination.
qualifier2	ViReal64	Currently unused, set to "0.0"

Accepted Values of *signal*

Connector Type	Possible Values of *signal*
Front Panel Aux I/O	0 = Disable **Veto:** 1 = Veto 2 = Switch Veto - TC890 3 = Inverted Veto 4 = Inverted Switch Veto - TC890 **Inputs:** **TC840/TC842**: 16 = arm, 17 = stop **TC890**: 1 = Bank switch, 2 = Marker **Outputs:** **TC840/TC842**: 48 = READY **TC890**: 32 = LVTTL low level, 33 = LVTTL high level

Return Value

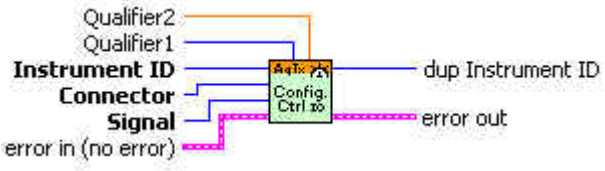
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_configControlIO (ViSession instrumentID, ViInt32
                    connector, ViInt32 signal,
                    ViInt32 qualifier1, ViReal64 qualifier2);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Configure Control I/O.vi
```



MATLAB MEX Representation

```
[status]= AqT3_configControlIO(instrumentID, connector, signal, qualifier1,
                    qualifier2)
```

2.3.111 AcqrsT3_configMemorySwitch

Purpose

Configures the memory bank switch triggering events. TC890 only.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
switchEnable	ViInt32	This is a bitfield to identify the unique event that can cause the switch = 1 switch on Aux I/O (use AcqrsT3_configControlIO to enable signal) = 2 switch on count of events on common channel = 4 switch on memory size limit
countEvent	ViInt32	number of events on the common channel
sizeMemory	ViInt32	memory size limit to use
reserved	ViInt32	Currently unused, set to "0"

Return Value

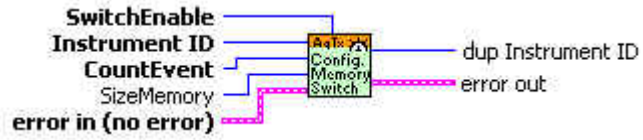
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_configMemorySwitch (ViSession instrumentID,
                    ViInt32 switchEnable, ViInt32 countEvent,
                    ViInt32 sizeMemory, ViInt32 reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Configure MemorySwitch.vi
```



MATLAB MEX Representation

```
[status] = AqT3_configMemorySwitch(instrumentID, switchEnable,    countEvent,
                    sizeMemory, reserved)
```

2.3.112 AcqrsT3_configMode

Purpose

Configures parameters for the operating mode of the instrument.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
mode	ViInt32	= 1 standard acquisition - the only TC840 and TC842 mode = 2 Time of Flight acquisition - the only TC890 mode
modifier	ViInt32	For TC840 and TC842 = 0 single acquisition = 1 multiple acquisitions
flags	ViInt32	= 0 internal reference clock = 1 external reference clock = 2 enable test signal

Return Value

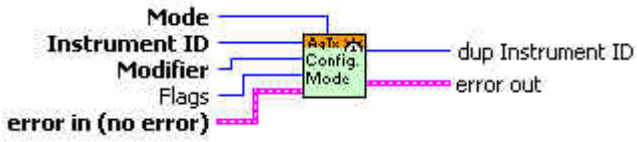
Name	Type	Description
status	ViStatus	Refer to Table 2-1 for error codes.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_configMode (ViSession instrumentID,
                    ViInt32 mode, ViInt32 modifier, ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Configure Mode.vi
```



MATLAB MEX Representation

```
[status]= AqT3_configMode(instrumentID, mode, modifier, flags)
```

2.3.113 AcqrsT3_forceTrig

Purpose

Generate a COMMON hit for a TC890.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
forceTrigType	ViInt32	Currently unused, set to “0”
modifier	ViInt32	Currently unused, set to “0”
flags	ViInt32	Currently unused, set to “0”

Return Value

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

Discussion

This function can be used to either

- measure times of multiple hits on the same or different channels, relative to a single origin. In this case, no signal would be connected on the 'COMMON' channel. Instead, 'AcqrsT3_forceTrig' would be called directly after 'AcqrsT3_acquire' to start the TC's real time counter. Subsequent hits on the other channels would then be measured relative to the moment 'forceTrig' was called.
- trigger a bank switch in 'Switch on event count' mode, by inserting additional 'dummy' COMMON hits after the last 'real' COMMON hit until the bank switch occurs.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_forceTrig(ViSession instrumentID,
         ViInt32 forceTrigType, ViInt32 modifier, ViInt32 flags);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Force Trigger.vi
```

MATLAB MEX Representation

```
[status]= AqT3_forceTrig(instrumentID, forceTrigType, modifier, flags)
```

2.3.114 AcqrsT3_getAcqConditions

Purpose

Returns the current acquisition parameters of the Time-to-Digital Converter.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
timeout	ViReal64	Timeout in seconds
flags	ViInt32	The LSB (bit 0) = 0 start timeout counter on Arm = 1 start timeout counter on first Common hit
reserved	ViInt32	Currently unused, set to "0"

Return Value

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

Discussion

See remarks under **AcqrsT3_configAcqConditions**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_getAcqConditions (ViSession instrumentID,
                            ViReal64* timeout,
                            ViInt32* flags, ViInt32* reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Query Acquisition Conditions.vi
```

MATLAB MEX Representation

```
[status timeoutP flagsP reservedP]= AqT3_getAcqConditions(instrumentID)
```

2.3.115 AcqrsT3_getChannel

Purpose

Returns the current channel parameters of the Time-to-Digital Converter.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	1…Nchan or −1 for the common channel −2 for the veto

Output

Name	Type	Description
mode	ViInt32	The LSB (bit 0) = 0 positive slope = 1 negative slope Bit 1 = 0 normal events = 1 pulse events with pulse type defined by the LSB (TC890 ONLY) The MSB (bit31) = 0 active channel = 1 inactive channel
level	ViReal64	Threshold value in Volts.
reserved	ViInt32	Currently unused, set to “0”

Return Value

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

Discussion

See remarks under **AcqrsT3_configChannel**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_getChannel (ViSession instrumentID,
                        ViInt32 channel, ViInt32* mode,
                         ViReal64* level,  ViInt32* reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Query Channel.vi
```

MATLAB MEX Representation

```
[status modeP levelP reservedP]= AqT3_getChannel(instrumentID, channel)
```

2.3.116 AcqrsT3_getControlIO

Purpose

Returns the current configuration of the auxiliary I/O connectors.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
connector	ViInt32	Connector Number 1 = Front Panel Aux I/O 1 2 = Front Panel Aux I/O 2

Output

Name	Type	Description
signal	ViInt32	See remarks under **AcqrsT3_configControlIO**
qualifier1	ViInt32	If the LSB (bit0) is set to 1 forces 50 Ohm termination for the connector
qualifier2	ViReal64	Currently unused

Return Value

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

Discussion

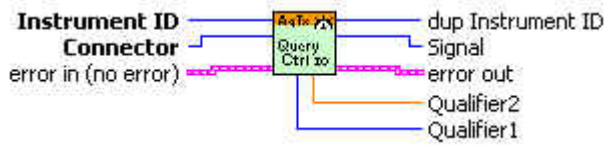
See remarks under **AcqrsT3_configControlIO**

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_getControlIO (ViSession instrumentID,
                        ViInt32 channel, ViInt32* signal,
                        ViInt32* qualiflier1, ViReal64* qualiflier2);
```

LabVIEW Representation

```
AcqrsT3 Query ControlIO.vi
```



MATLAB MEX Representation

```
[status signal qualifier1 qualifier2]= AqT3_getControlIO(instrumentID,
                        connector)
```

2.3.117 AcqrsT3_getMemorySwitch

Purpose

Returns the current channel parameters of the memory bank switch operation.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
switchEnable	ViInt32	This is a bitfield to identify the enabled events = 1 switch on I/O Aux = 2 switch on count of events on common channel = 4 switch on memory size limit
countEvent	ViInt32	number of events on the common channel
sizeMemory	ViInt32	memory size limit to use
reserved	ViInt32	Currently unused

Return Value

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

Discussion

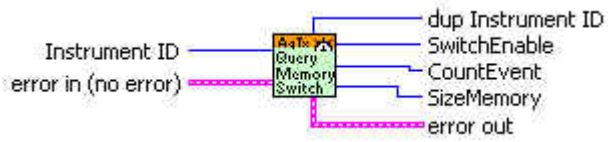
See remarks under **AcqrsT3_configMemorySwitch**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_getMemorySwitch (ViSession instrumentID,
                      ViInt32* switchEnable, ViInt32* countEvent,
                       ViInt32* sizeMemory, ViInt32* reserved);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Query MemorySwitch.vi
```



MATLAB MEX Representation

```
[status switchEnableP countEventP sizeMemoryP reservedP]=
                     AqT3_getMemorySwitch(instrumentID)
```

2.3.118 AcqrsT3_getMode

Purpose

Returns the current operational mode of the Time-to-Digital Converter.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier

Output

Name	Type	Description
mode	ViInt32	= 1 standard acquisition TC840 and TC842 = 2 Time of Flight acquisition TC890
modifier	ViInt32	For TC840 and TC842 = 0 single hit = 1 multiple hits
flags	ViInt32	= 0 internal reference clock = 1 external reference clock

Return Value

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

Discussion

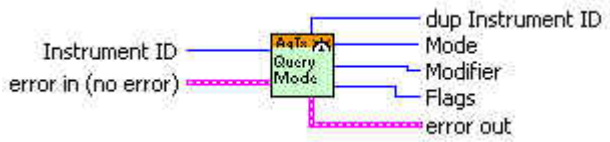
See remarks under **AcqrsT3_configMode**.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_getMode (ViSession instrumentID,
                        ViInt32* mode,
                         ViInt32* modifier, ViInt32* flags);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Query Mode.vi
```



MATLAB MEX Representation

```
[status mode modifiers flags] = AqT3_getMode(instrumentID)
```

2.3.119 AcqrsT3_readData

Purpose

Returns all Time-to-Digital Converter information. The sample data is returned in a model dependent form and as specified in the **AqT3ReadParameters** structure.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	Reserved for future use (must be set to 0)
readPar	AqT3ReadParameters	Requested parameters for the acquired data.

Output

Name	Type	Description
dataDesc	AqT3DataDescriptor	Data descriptor structure needed for interpretation

Return Value

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

Read Parameters in AqT3ReadParameters

Name	Type	Description
dataArray	ViAddr	User-allocated time value data buffer .
dataSizeInBytes	ViUInt32	Number of bytes in the user-allocated *dataArray*. Used for verification / protection. See discussion for required size.
nbrSamples	ViInt32	Number of samples requested. For the TC890 it is used for the maximum number of 4-byte structures to be returned by the read (see dataType = 4 discussion below)
dataType	ViInt32	Type representation of the data 4 = ReadRawData = raw format = 4 bytes as used for the TC890 TOF mode 3 = ReadReal64 = 64-bit (double) = 8 bytes 2 = ReadInt32 = 32-bit (integer) = 4 bytes TC840 only
readMode	ViInt32	0 = AqT3ReadStandard = standard readout mode 1 = AqT3ReadContinuous = TOF mode - TC890 only
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Data Descriptor AqT3DataDescriptor

Name	Type	Description
dataPtr	ViAddr	Pointer to time value data buffer. May differ from dataArray above!
nbrSamples	ViInt32	number of samples returned
sampleSize	ViInt32	Size in bytes of the time data format in use
sampleType	ViInt32	type of the returned samples, see AqT3SampleType
flags	ViInt32	For TC890 ONLY Bit 0: Internal memory overflow flag Bit 1: External memory overflow flag
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Discussion

All structures used in this function can be found in the header files **AcqirisT3Interface.h** and **AcqirisDataTypes.h**.

The type of the **dataArray** is determined from the **AqT3ReadParameters** struct entry **dataType**.

- **dataType** = 4 is used for raw data. For example, the 32-bit natural readout of the TC890 TOF multihit mode is of **AqT3SampleType** AqT3Struct50ps6ch and has the following format:

31	28-30	0-27
Overflow	Channel	Data

where

Channel = 1…6 denotes the physical channels. The Data bits give the time value in units of 50 ps

0 is for the start of the next event. In this case the Data bits give the count of the common start within the current acquisition

7 is for marker data with Data

= 0 : Switch from Auxiliary input A
= 1 : Switch marker: Common channel Event count.
= 2 : Switch marker: Memory Full.
= 16 : Marker: Auxiliary input B marker.

- **dataType** = 3 is used for double floating-point format time results. These results are always in seconds. A value of 1e10 is a sign that the channel in question did not see a stop.

- **dataType** = 2 is used for integer format time results. These results are always in multiples of the granularity given by the **AqT3SampleType** value of AqT3Count50psInt32. A value of 0 is a sign that the channel in question did not see a stop.

⚠ The **dataSizeInBytes** must fulfill the storage requirement for the raw data read from the device. This means that for the TC840/TC842

single hit mode - 104 bytes for TC840, 416 bytes for TC842

multi-start mode - 52KB = 53248 bytes

and for the TC890 you must configure it as a function of the number of expected values, including the start, and markers counting 4 bytes for each. The worst case is the full bank of 8MB = 8388608.

Data beyond the point implied by the **nbrSamples** returned value must be ignored.

The TC890 memory overflow flags show whether that condition happened since the previous call of the readData routine.

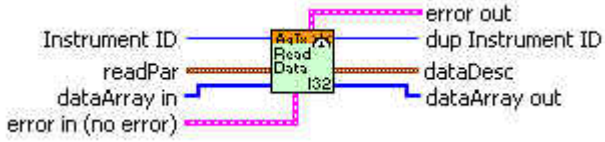
LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_readData(ViSession instrumentID,
                    ViInt32 channel, AqT3ReadParameters* readPar,
                    AqT3DataDescriptor* dataDesc);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Read Data.vi
```

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



MATLAB MEX Representation

```
[status dataDesc dataArray] = AqT3_readData(instrumentID, channel, readPar)
```

2.3.120 AcqrsT3_readDataInt32

Purpose

Returns all Time-to-Digital Converter information for a TC840 or TC890. The sample data is returned in a model dependent form and as specified in the **AqT3ReadParameters** structure.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	Reserved for future use (must be set to 0)
readPar	AqT3ReadParameters	Requested parameters for the acquired data.

Output

Name	Type	Description
dataArrayP	ViInt32*	Data array pointer
dataDesc	AqT3DataDescriptor	Data descriptor structure needed for interpretation

Return Value

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

Read Parameters in AqT3ReadParameters

Name	Type	Description
dataArray	ViAddr	Unused - set to NULL .
dataSizeInBytes	ViUInt32	Number of bytes in the user-allocated *dataArray*. Used for verification / protection. See discussion for required size.
nbrSamples	ViInt32	Number of samples requested. For the TC890 it is used for the maximum number of 4-byte structures to be returned by the read (see dataType = 4 discussion below)
dataType	ViInt32	Type representation of the data 4 = ReadRawData = raw format = 4 bytes as used for the TC890 TOF mode 2 = ReadInt32 = 32-bit (integer) = 4 bytes
readMode	ViInt32	0 = AqT3ReadStandard = standard readout mode 1 = AqT3ReadContinuous = TOF mode - TC890 only
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Data Descriptor AqT3DataDescriptor

Name	Type	Description
dataPtr	ViInt32	Not relevant in this context and should be ignored
nbrSamples	ViInt32	number of samples returned
sampleSize	ViInt32	Size in bytes of the time data format in use
sampleType	ViInt32	type of the returned samples, see AqT3SampleType
flags	ViInt32	For TC890 ONLY Bit 0: Internal memory overflow flag Bit 1: External memory overflow flag
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Discussion

All structures used in this function can be found in the header files **AcqirisT3Interface.h** and **AcqirisDataTypes.h**.

The type of the **dataArray** is determined from the **AqT3ReadParameters** struct entry **dataType**.

- **dataType** = 4 is used for raw data. For example, the 32-bit natural readout of the TC890 TOF multihit mode is of **AqT3SampleType** AqT3Struct50ps6ch and has the following format:

31	28-30	0-27
Overflow	Channel	Data

where

Channel = 1…6 denotes the physical channels. The Data bits give the time value in units of 50 ps

0 is for the start of the next event. In this case the Data bits give the count of the common start within the current acquisition

7 is for marker data with Data

= 0 : Switch from Auxiliary input A
= 1 : Switch marker: Common channel Event count.
= 2 : Switch marker: Memory Full.
= 16 : Marker: Auxiliary input B marker.

- **dataType** = 2 is used for integer format time results. These results are always in multiples of the granularity given by the **AqT3SampleType** value of AqT3Count50psInt32.

⚠ The **dataSizeInBytes** must fulfill the storage requirement for the raw data read from the device. This means that for the TC840

single hit mode - 104 bytes

multi-start mode - 52KB = 53248 bytes

and for the TC890 you must configure it as a function of the number of expected values, including the start, and markers counting 4 bytes for each. The worst case is the full bank of 8MB = 8388608.

Data beyond the point implied by the **nbrSamples** returned value must be ignored.

The TC890 memory overflow flags show whether that condition happened since the previous call of the readData routine.

The allocated data array must be 32-bit aligned. If it is not an error status will be generated.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_readDataInt32(ViSession instrumentID,
                     ViInt32 channel, AqT3ReadParameters* readPar,
                     ViInt32* dataArrayP, AqT3DataDescriptor* dataDesc);
```

LabVIEW Representation

```
Use the polymorphic Acqiris Tx.lvlib: (or Aq Tx) Read Data.vi
```

MATLAB MEX Representation

```
[status dataDesc dataArray] = AqT3_readData(instrumentID, channel, readPar)
```

2.3.121 AcqrsT3_readDataReal64

Purpose

Returns all Time-to-Digital Converter information. The sample data is returned in a model dependent form and as specified in the **AqT3ReadParameters** structure.

Parameters

Input

Name	Type	Description
instrumentID	ViSession	Instrument identifier
channel	ViInt32	Reserved for future use (must be set to 0)
readPar	AqT3ReadParameters	Requested parameters for the acquired data.

Output

Name	Type	Description
dataArrayP	ViReal64*	Data array pointer
dataDesc	AqT3DataDescriptor	Data descriptor structure needed for interpretation

Return Value

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

Read Parameters in AqT3ReadParameters

Name	Type	Description
dataArray	ViAddr	Unused - set to NULL .
dataSizeInBytes	ViUInt32	Number of bytes in the user-allocated *dataArray*. Used for verification / protection. See discussion for required size.
nbrSamples	ViInt32	Number of samples requested. For the TC890 it is used for the maximum number of 4-byte structures to be returned by the read (see dataType = 4 discussion below)
dataType	ViInt32	Type representation of the data 3 = ReadReal64 = 64-bit (double) = 8 bytes
readMode	ViInt32	0 = AqT3ReadStandard = standard readout mode
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Data Descriptor AqT3DataDescriptor

Name	Type	Description
dataPtr	ViAddr	Not relevant in this context and should be ignored
nbrSamples	ViInt32	number of samples returned
sampleSize	ViInt32	Size in bytes of the time data format in use
sampleType	ViInt32	type of the returned samples, see AqT3SampleType
flags	ViInt32	Unused
reserved3	ViInt32	Reserved for future use
reserved2	ViInt32	Reserved for future use
reserved1	ViInt32	Reserved for future use

Discussion

All structures used in this function can be found in the header files **AcqirisT3Interface.h** and **AcqirisDataTypes.h**.

The type of the **dataArray** is determined from the **AqT3ReadParameters** struct entry **dataType**.

- **dataType** = 3 is used for double floating-point format time results. These results are always in seconds.

⚠ The **dataSizeInBytes** must fulfill the storage requirement for the raw data read from the device. This means that for the TC840/TC842

single hit mode - 104 bytes for TC840, 416 bytes for TC842

multi-start mode - 52KB = 53248 bytes

and for the TC890 you must configure it as a function of the number of expected values, including the start, and markers counting 4 bytes for each. The worst case is the full bank of 8MB = 8388608.

Data beyond the point implied by the **nbrSamples** returned value must be ignored.

The TC890 memory overflow flags show whether that condition happened since the previous call of the readData routine.

The allocated data array must be 32-bit aligned. If it is not an error status will be generated.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_readDataReal64(ViSession instrumentID,
                    ViInt32 channel, AqT3ReadParameters* readPar,
                    ViReal64* dataArrayP, AqT3DataDescriptor* dataDesc);
```

LabVIEW Representation

```
Use the polymorphic Acqiris Tx.lvlib: (or Aq Tx) Read Data.vi
```

MATLAB MEX Representation

```
[status dataDesc dataArray] = AqT3_readData(instrumentID, channel, readPar)
```

2.3.122 AcqrsT3_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.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_stopAcquisition(ViSession instrumentID);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Stop Acquisition.vi
```

MATLAB MEX Representation

```
[status] = AqT3_stopAcquisition(instrumentID)
```

2.3.123 AcqrsT3_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 comes first. 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.

LabWindowsCVI/Visual C++ Representation

```
ViStatus status = AcqrsT3_waitForEndOfAcquisition (ViSession instrumentID,
                    ViInt32 timeout);
```

LabVIEW Representation

```
Acqiris Tx.lvlib: (or Aq Tx) Wait For End Of Acquisition.vi
```

MATLAB MEX Representation

```
[status] = AqT3_waitForEndOfAcquisition(instrumentID, timeOut)
```